EEN1083

Data analysis and machine learning I

Ali Intizar

DCU Ollscoil Chathair Bhaile Átha Cliath Dublin City University

Regularization

DCU Ollscoil Chathair Bhaile Átha Cliath Dublin City University

Outline

Norms

Overfitting

Regularization

L2 regularization

L1 regularization

Norms

In mathematics, a norm is a function from a real or complex vector space to the non-negative real numbers that behaves in certain ways like the distance from the origin: it commutes with scaling, obeys a form of the triangle inequality, and is zero only at the origin. (Wikipedia)

In machine learning norms are a way to measure the size of a vector, matrix or tensor

Norms

Most common norm is the Euclidean norm or L_2 norm, denoted $\|\mathbf{x}\|_2$ or just $\|\mathbf{x}\|$. Gives the length of the vector.

$$\|\mathbf{x}\| = \sqrt{\mathbf{x}^T\mathbf{x}} = \sqrt{\sum_i x_i^2}$$

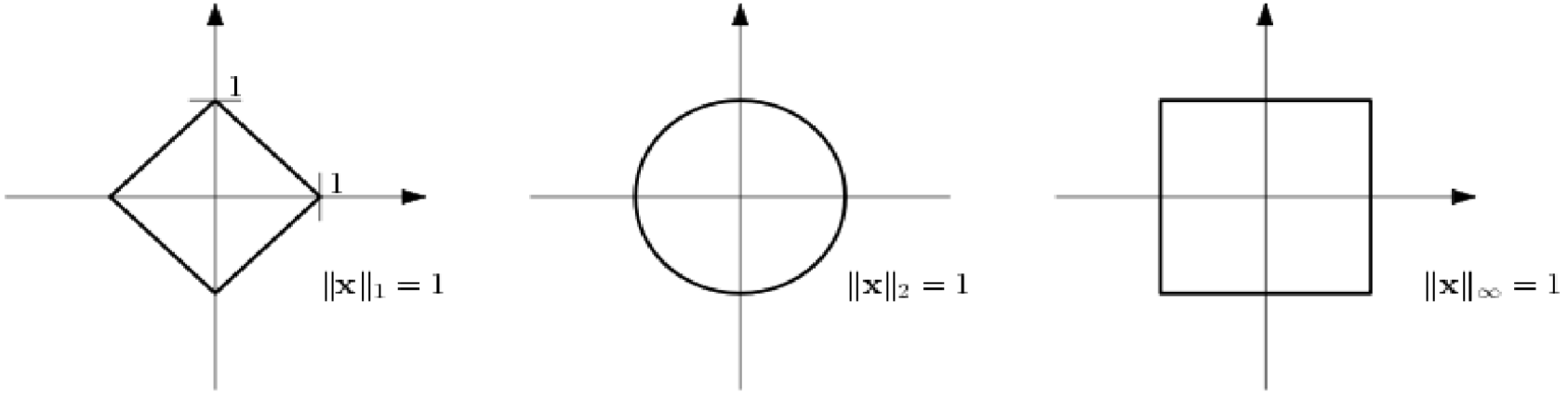
The squared Euclidean norm $\|\mathbf{x}\|_2^2$ is often useful:

$$\|\mathbf{x}\|^2 = \mathbf{x}^T\mathbf{x} = \mathbf{x} \cdot \mathbf{x}$$

Another common norm is the L_1 norm, which is the sum of absolute values of $\mathbf{x}$

$$\|\mathbf{x}\|_1 = \sum_i |x_i|$$

Norms



Properties of norms

A norm is any function p from vectors to $\mathbb{R}$ that satisfies:

1. $p(\alpha \mathbf{x}) = |\alpha| p(\mathbf{x})$ for $\alpha \in \mathbb{R}$
2. $p(\mathbf{x} + \mathbf{y}) \leq p(\mathbf{x}) + p(\mathbf{y})$ (triangle inequality)
3. $p(\mathbf{x}) = 0 \iff \mathbf{x} = \mathbf{0}$

E.g. $\|5\mathbf{x}\| = 5\|\mathbf{x}\|$ for any norm $\|\cdot\|$.
We also have $p(-\mathbf{x}) = p(\mathbf{x})$

Norms and distance metrics

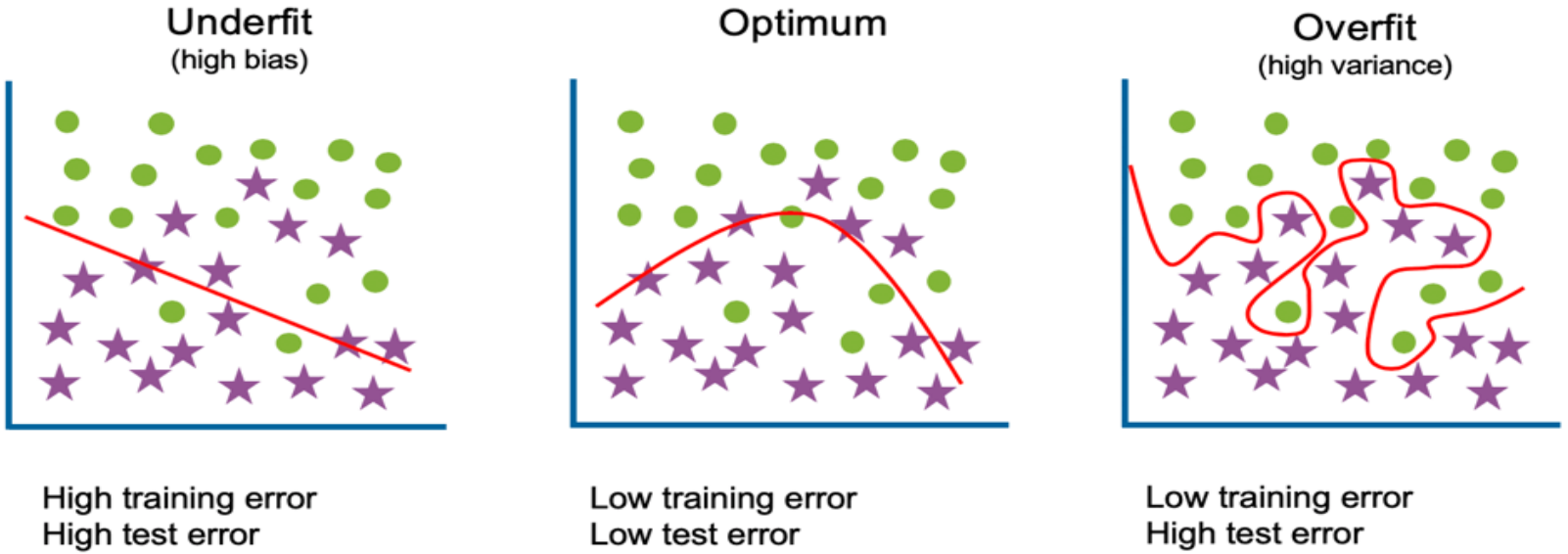
Any vector space V and norm p can be used to induce a distance metric (define a **metric space**) by defining the distance metric to be $d(\mathbf{x}, \mathbf{y}) = p(\mathbf{x} - \mathbf{y})$

E.g. The space of D dimensional real valued vectors $\mathbb{R}^D$ and the L_2 norm give the Euclidean space with metric $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$.

In 2D this gives the familiar Euclidean distance between two points $\mathbf{x}$ and $\mathbf{y}$:

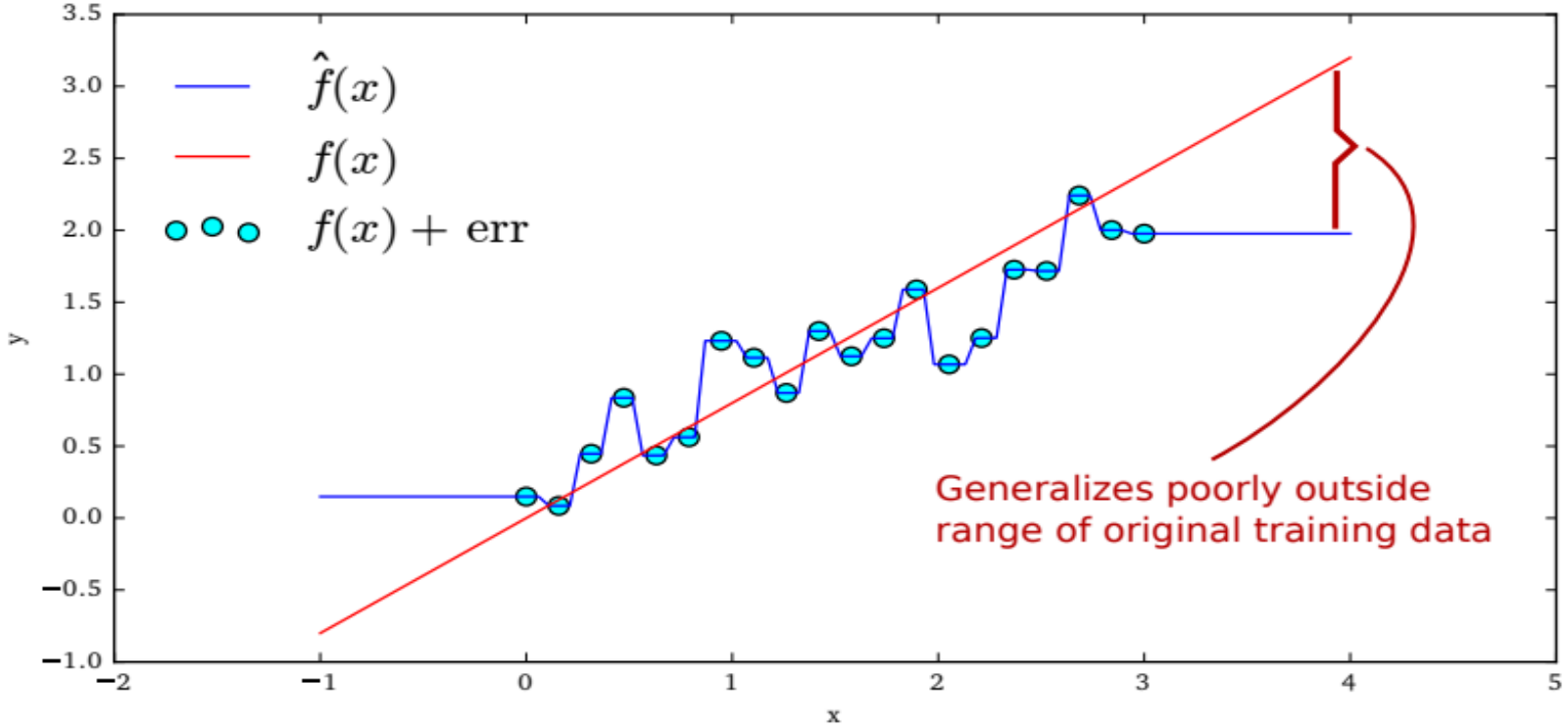
$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

Overfitting



https://www.javatpoint.com/overfitting-in-machine-learning

Nearest neighbours generalisation

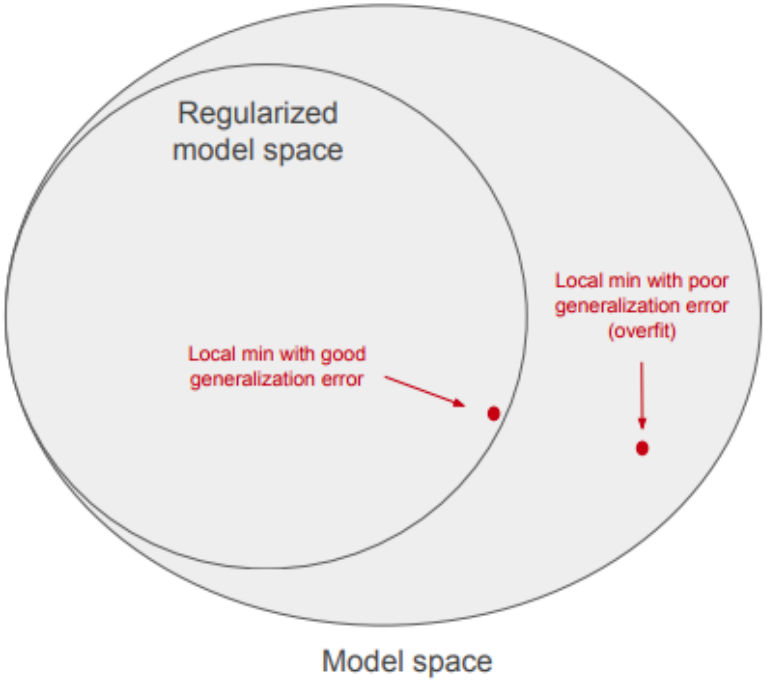


Structural risk minimization

•**Structural risk minimization:** prevent overfitting by balancing model complexity with success at fitting training data.

• Idea: Given two models (hypotheses) with similar training error, prefer ones with lower complexity.

• **Regularization**: add additional information to solve an ill-posed problem or prevent overfitting. Additional information is often a **penalty** included in the loss function.



L2 regularization

Add a penalty to the loss function for large weights.

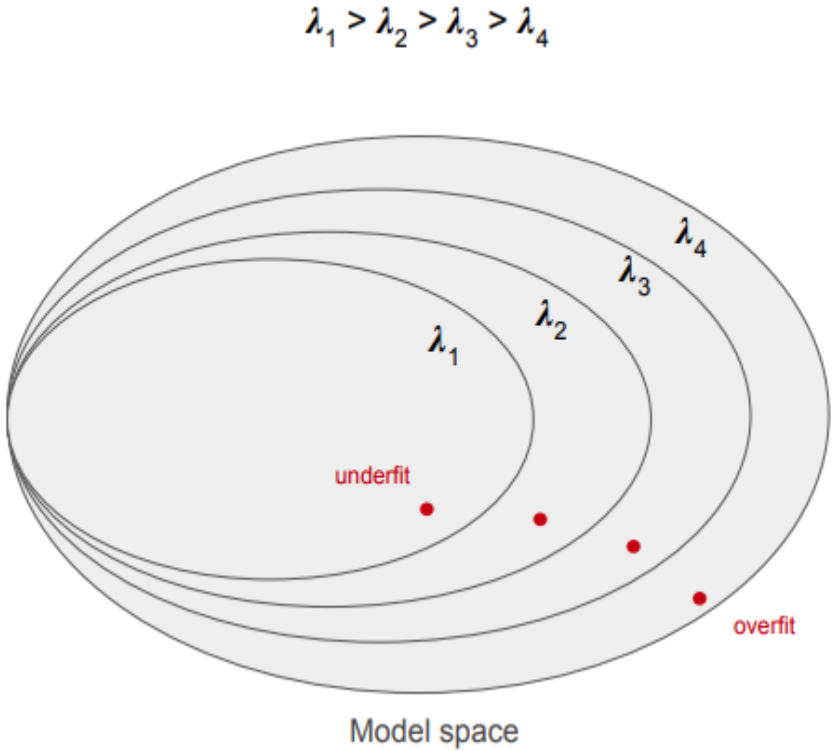
Penalty is on the L_2 norm of the weights

$$\|\mathbf{w}\|_2^2 = \mathbf{w}^T\mathbf{w} = \sum_{i=1}^{D} w_i^2$$

The modified loss is:

$$\mathcal{L} = \mathcal{L}_{\text{data}} + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

where λ is the **regularization parameter**, which controls the strength of the regularization



L2 regularization and gradient descent

L_2 regularized loss:

$$\mathcal{L} = \mathcal{L}_{\text{data}} + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

Taking the gradient wrt. **w** we get

$$\nabla_{\mathbf{w}}\mathcal{L} = \nabla_{\mathbf{w}}\mathcal{L}_{\text{data}} + \lambda\mathbf{w}$$

which gives the gradient descent update rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha\nabla_{\mathbf{w}}\mathcal{L}_{\text{data}} - \alpha\lambda\mathbf{w}_t$$

This can be understood as slowly *decaying* the weights toward zero with each iteration. In the neural networks literature L_2 regularization is known as **weight decay**.

Why the L2 penalty is reasonable

Imagine we had a training set for which the first two features are always equal $x_1 = x_2$. E.g. this could happen due to a broken sensor, or an image with a watermark.

	x_1	x_2	x_3	x_4	...	x_N	y
$\mathbf{x}_1$	1	1	4	7	...	2	0
$\mathbf{x}_2$	1	1	2	1	...	6	1
$\mathbf{x}_3$	1	1	9	2	...	5	1
:	:	:	:	:	:	:	⋱

If the decision function $f(x) = \mathbf{w}^T\mathbf{x} + b = w_1x_1 + w_2x_2 + \ldots + w_Nx_N + b$, then we can set $w_1 = -w_2$ to any constant we want without changing the outcome or the loss.

Why the L2 penalty is reasonable

Lets say we choose $w_1 = 1000, w_2 = -1000$.

We observe a test point in which $x_2 = 0$. Now we have $f(x) = 1000x_1 + 1000(0) + \ldots$, which changes the value of the decision function by +1000!

With large weights, small changes in input can produce large changes in output!

If we chose $w_1 = w_2 = 0$, the error on the training set would be the same and there would have been no such problem with the test point.

The L_2 penalty encourages finding solutions with **smaller weights**.

Ridge regression

Applying the L_2 penalty to linear regression gives a model called **ridge regression**. The ridge regression loss is:

$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{N}(y - f(\mathbf{x}))^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

The optimal parameters are the solution to

$$\arg\min_{\mathbf{w}} L$$

As with linear regression, this can be solved in closed form by taking derivatives and setting to zero. The solution turns out to be:

$$\hat{\mathbf{w}} = (X^T X + \lambda I)^{-1} X^T \mathbf{y}$$

Choosing the regularization parameter

The regularization parameter λ controls the tradeoff between having small weights and fitting the training data.

- Smaller values for λ place more emphasis on fitting the training data and may be more suitable in low-noise settings or when there are few parameters to fit.
- Larger values for λ place more emphasis on making the values of $\mathbf{w}$ small, and are more suitable when there is a lot of noise or many parameters to fit.

As usual, the best way to choose an appropriate value for λ is using model selection procedures like a hold-out validation set or cross-validation.

Notes on L2 regularization

L_2 regularization is also known as **Tikhonov regularization**.

L_2 regularization in ridge regression corresponds to a Gaussian prior on the weights with zero mean and variance proportional to $1/\lambda$.

Can be used to find least squares solutions to ill-posed problems:

- more free variables than constraints,
- $(X^T X)$ is singular,
- $(X^T X + \lambda I)$ is invertible.

Essential for overparameterized problems like softmax regression.

L1 regularization

L_2 regularization encourages solutions with small weights.

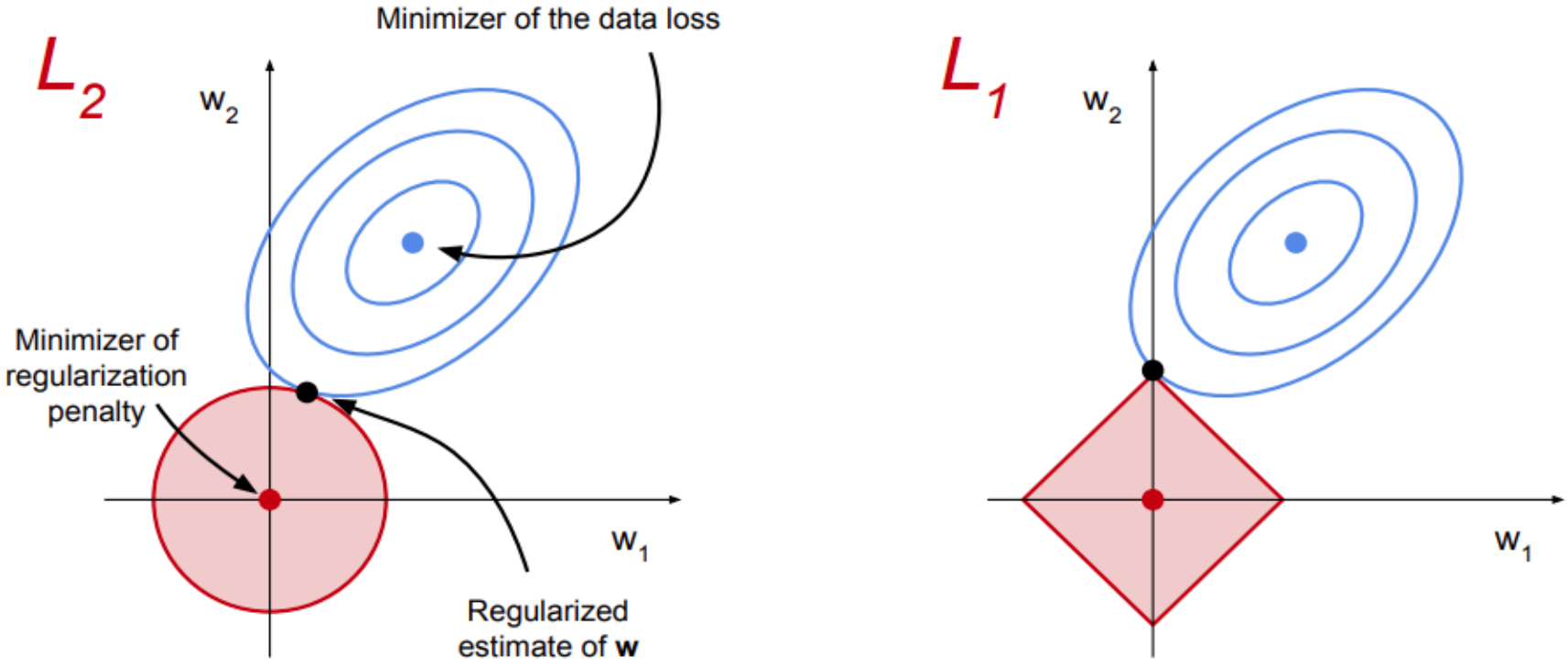
Sometimes you want to find solutions in which most of the weights are exactly zero. These are called **sparse** solutions.

E.g. **feature selection**: you have many features, and you want your decision function to only use a subset. Any feature with weight zero is not used.

L_1 regularization encourages sparse solutions by using penalizing the L_1-norm of the weights.

$$\|\mathbf{w}\|_1 = \sum_{i=1}^{D} |w_i|$$

How does L1 regularization encourage sparsity?



(Sub)-gradient descent for L1 regularization

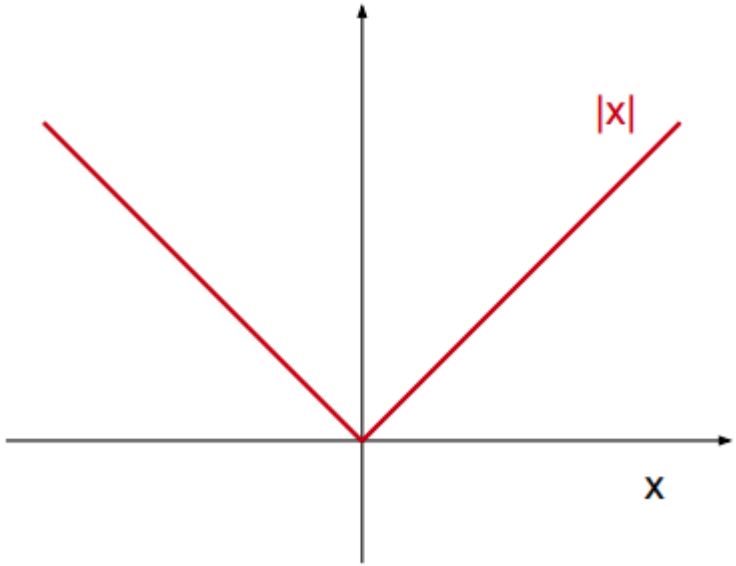
L_1 regularized loss:

$$\mathcal{L} = \mathcal{L}_{\text{data}} + \lambda \|\mathbf{w}\|_1$$

$$= \mathcal{L}_{\text{data}} + \lambda \sum_{i=1}^{D} |w_i|$$

The absolute value has a discontinuity at zero so gradient is not defined everywhere.

For gradient descent we can use a **subgradient**.



(Sub)-gradient descent for L1 regularization

A subgradient for $|w|$ is:

$$\nabla_w |w| = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \\ 0 & w = 0 \end{cases}$$

which can be written more succinctly using the sign function $\nabla_w |w| = \text{sign}(w)$.

This gives us the subgradient descent update rule for L_1 regularization:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla_{\mathbf{w}} \mathcal{L} - \lambda \sum_{i=0}^{D} \text{sign}(w_i)$$

The LASSO and Elastic Net

Combining ordinary least squares regression with the L_1 penalty gives rise to the so called **LASSO** (least absolute shrinkage and selection operator).

Can solve using subgradient descent as shown. Generally preferable to use a faster algorithm like **least angle regression** (LARS).

It is also possible to use **both** L_1 and L_2 regularizers together. This model is called the **elastic net**.

In scikit-learn

- sklearn.linear model.Ridge
- sklearn.linear model.Lasso
- sklearn.linear model.LassoLars
- sklearn.linear model.ElasticNet

Logistic regression in scikit-learn can also take an L_1 or L_2 regularization penalty: LogisticRegression(penalty='l2', C=1.0), where $C > 0$ is the **inverse** of λ (smaller values specify stronger regularization).

Scikit-learn also has classes that do automatic (and optimized) cross validation to figure out the best value for the regularization parameter(s). E.g.:

- sklearn.linear model.RidgeCV
- sklearn.linear model.LassoCV

Resources

Stanford machine learning lectures (Andrew Ng):

- Lecture 3: linear and logistic regression
  http://www.youtube.com/watch?v=HZ4cvaztQEs
- Lecture 4: Generalized linear models
  http://www.youtube.com/watch?v=nLKOQfKLUks

Caltech machine learning lectures (Yaser Abu-Mostafa):

- Lecture 12: Regularization
  http://www.youtube.com/watch?v=I-VfYXzC5ro

Resources

Oxford deep learning lectures (Nando de Freitas):

- Lecture 4: Regularization 1
  http://www.youtube.com/watch?v=VR0W_PNwLGw
- Lecture 5: Regularization 2
  http://www.youtube.com/watch?v=VR0W_PNwLGw