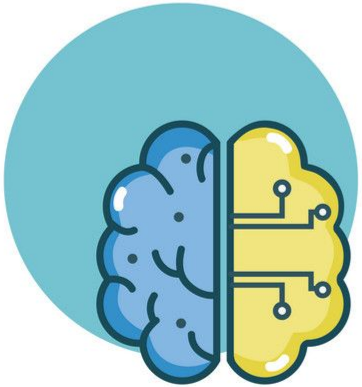
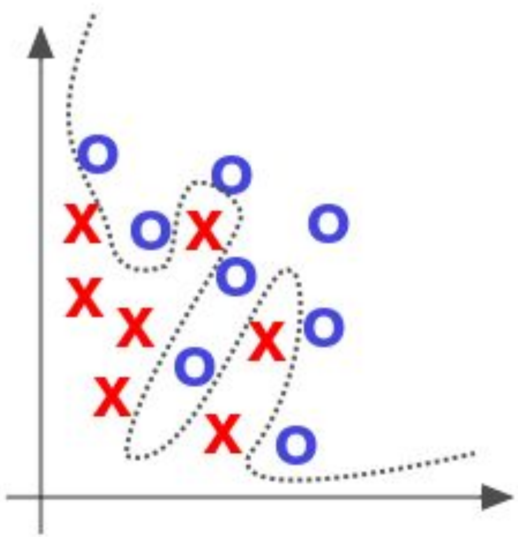
INTRODUCTION TO MACHINE LEARNING

REGULARIZATION

Elisa Ricci

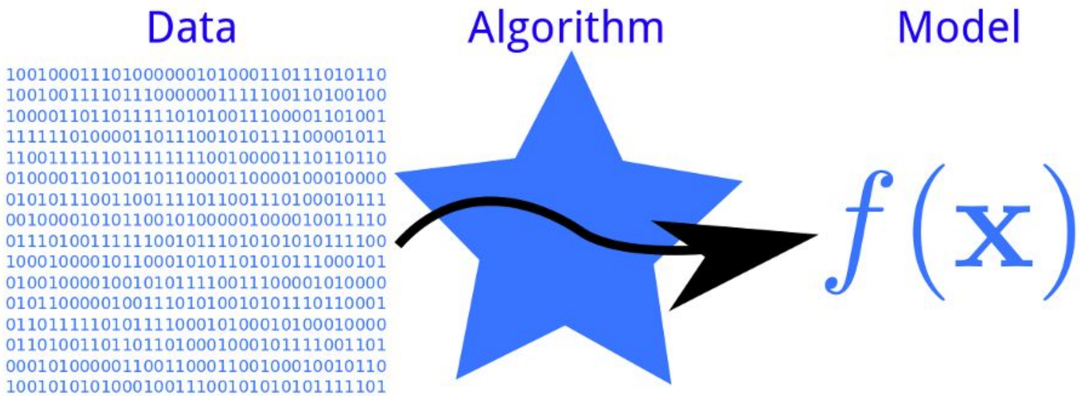




Avoid Overfitting

Machine Learning Idea

- ML allows computers to acquire knowledge.
- Knowledge is acquired through **algorithms** by learning and inferring from **data**.
- Knowledge is represented by a **model**.
- The model is used on future data.



Linear models

- Perceptron algorithm is one example of a linear classifier
- Many, many other algorithms learn a line (i.e. a setting of a linear combination of weights)
- Goals:
  - Explore a number of linear training algorithms

Model-based machine learning

1. Pick a **model**
   - e.g. a hyperplane, a decision tree,…
   - A model is defined by a collection of parameters
2. Pick a criterion to optimize (aka **objective function**)
   - e.g. training error
3. Develop a **learning algorithm**
   - the algorithm should try and minimize the criteria, sometimes in a heuristic way (i.e. non-optimally), sometimes exactly

Model-based machine learning

1. Pick a model

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

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

$$\sum_{i=1}^{n} 1\left[y_i(w \cdot x_i + b) \le 0\right]$$

3. Develop a learning algorithm

$$\operatorname{argmin}_{w,b} \sum_{i=1}^{n} 1\left[y_i(w \cdot x_i + b) \le 0\right]$$

Find w and b that minimize the 0/1 loss (i.e. training error)

Model-based machine learning

1. pick a model

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

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

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

use a convex surrogate loss function

3. develop a learning algorithm

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

Find w and b that minimize the surrogate loss

Gradient descent

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

$$w_j = w_j - \frac{d}{dw_i} loss(w)$$

Why negative?

Gradient descent

- For our choice of the loss we have:

$$w_j = w_j - \eta \frac{d}{dw_i} loss(w)$$

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

Exponential update rule

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

---

for each example x_i:

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

Does this look familiar?

Perceptron learning algorithm!

repeat until convergence (or for some # of iterations):

for each training example (f_1, f_2, …, f_n, label):

$$prediction = b + \sum_{i=1}^{n} w_i f_i$$

if *prediction* * *label* ≤ 0: // they don't agree

for each w_i:

$w_i = w_i + f_i$*label

$b = b$ + label

Note: for gradient descent, we always update

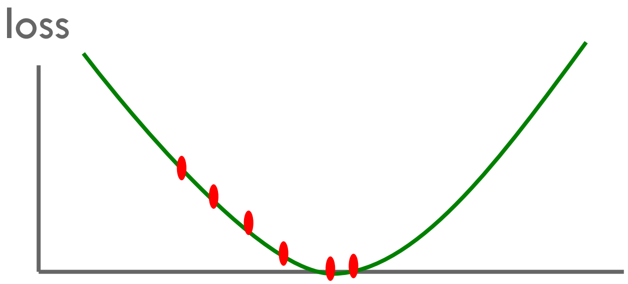
$$w_j = w_j + \eta y_i x_{ij} \exp(-y_i(w \cdot x_i + b))$$

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

One concern

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

We are calculating this on the **training set**



We still need to be careful about **overfitting**!

The $\min_{w,b} Loss$ on the training set is generally NOT the min for the test set

How did we deal with this?

REGULARIZATION

- A regularizer is an additional criterion to the loss function to make sure that we do not overfit
- It is called a regularizer since it tries to keep the parameters more normal/regular
- It is a bias on the model that forces the learning to prefer certain types of weights over others

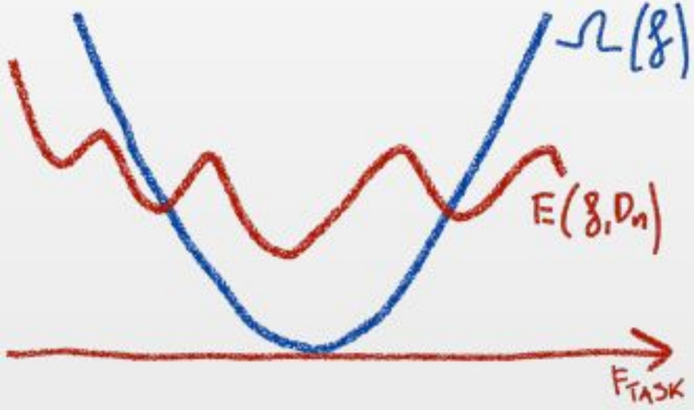
$$\operatorname{argmin}_{w,b} \sum_{i=1}^{n} loss(yy') + \lambda \; regularizer(w, b)$$

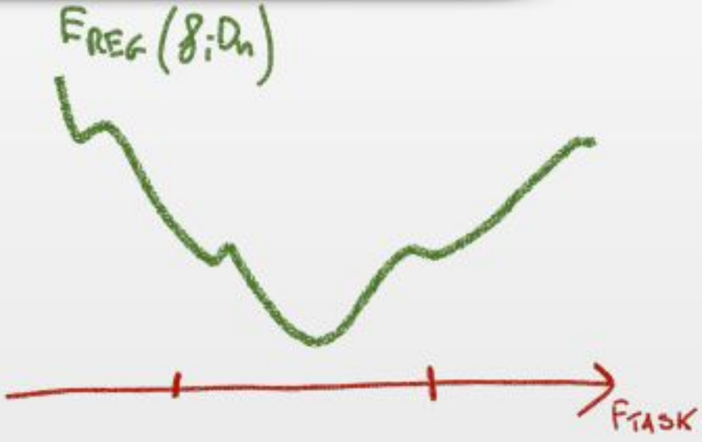
So Far...

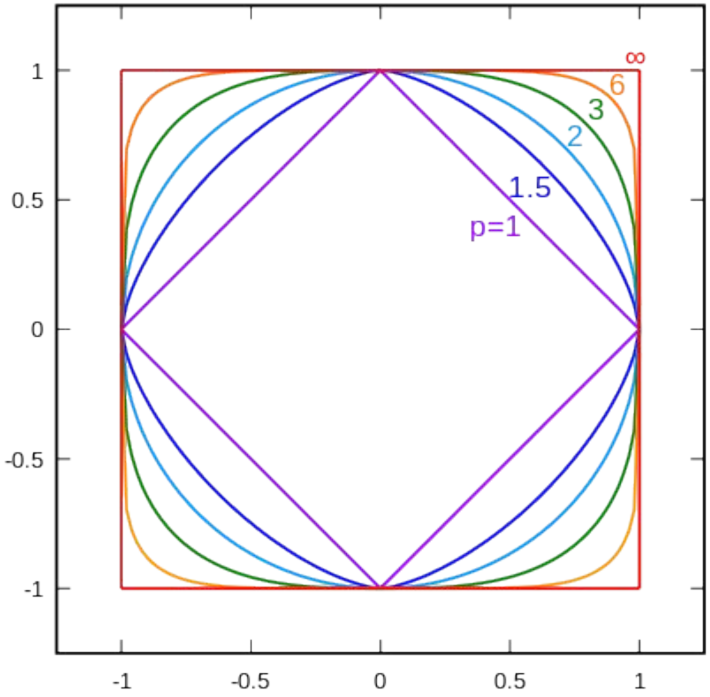
Modification of the training error function with a term $\Omega(f)$ that typically penalizes complex solutions

TRADE-OFF PARAMETER

$$\boxed{E_{\text{reg}}(f;\mathscr{D}_n) = E(f;\mathscr{D}_n) + \lambda_n \Omega(f)}$$







Regularizers

REGULARIZATION

- A regularizer is an additional criterion to the loss function to make sure that we do not overfit
- It is called a regularizer since it tries to keep the parameters more normal/regular
- It is a bias on the model that 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

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

Should we allow all possible weights?

Any preferences?

What makes for a “simpler” model for a linear model?

Regularizers

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

- Generally, we do not want huge weights: if weights are large, a small change in a feature can result in a large change in the prediction
- Might also prefer weights of 0 for features that are not useful

Regularizers

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

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

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

Common regularizers

sum of the weights $$r(w,b) = \sum \left| w_j \right|$$

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

What's the difference between these?

Common regularizers

sum of the weights

$$r(w,b) = \sum |w_j|$$

sum of the squared weights

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

Squared weights penalizes large values more

Sum of weights will penalize small values more

P-NORM

sum of the weights (1-norm) $$r(w,b) = \sum |w_j|$$

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

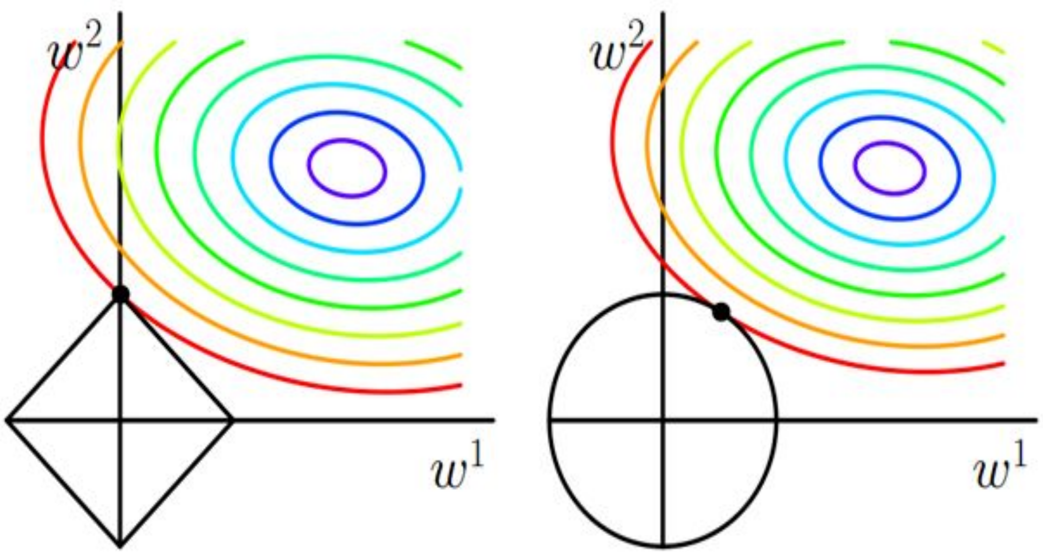
---

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

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

Larger values of p discourage large weights more

L1/L2-norms visualized

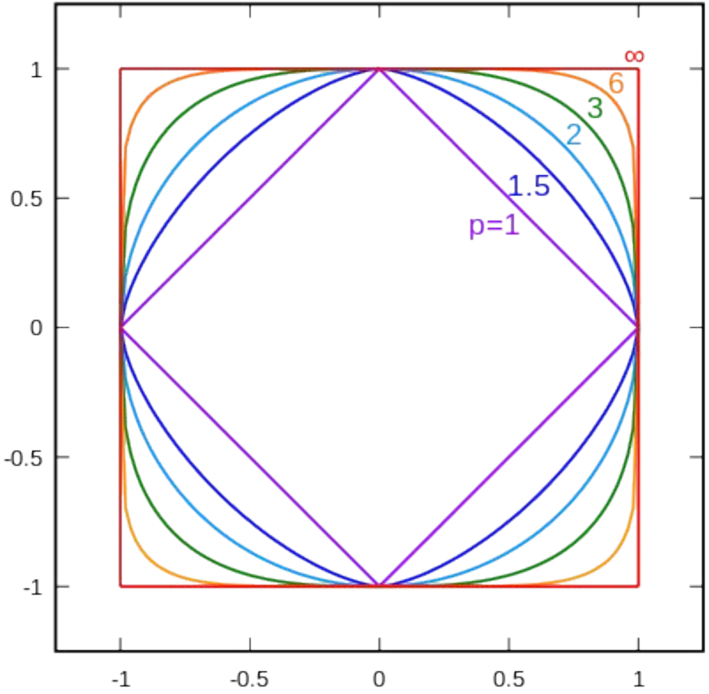


(a) ℓ_1-**ball meets quadratic function.** ℓ_1-ball has corners. It's very likely that the meet-point is at one of the corners.

(b) ℓ_2-**ball meets quadratic function.** ℓ_2-ball has no corner. It is very unlikely that the meet-point is on any of axes."

https://zhuanlan.zhihu.com/p/28023308

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

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

Find w and b that minimize

Minimizing with a regularizer

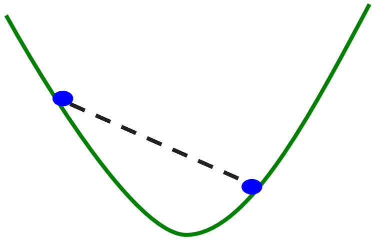
We know how to solve convex minimization problems using gradient descent:

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

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

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

Convexity



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

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:

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

p-norms are convex

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

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

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

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 any dimension:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)

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

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

Some more maths

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

$\vdots$ (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 any dimension:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)

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

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

The update

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

learning rate

direction to update

constant: how far from wrong

regularization

What effect does the regularizer have?

The update

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

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

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

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

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

The update

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

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:

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

Model-based machine learning

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

Is gradient descent the only way to find w and b?

No! Many other ways to find the minimum.
Some don't even require iteration
Whole field called convex optimization

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

Many Methods

- (Ordinary) Least squares: squared loss
- Ridge regression: squared loss with L2 regularization
- Lasso regression: squared loss with L1 regularization
- Elastic regression: squared loss with L1 AND L2 regularization
- Logistic regression: logistic loss

QUESTIONS?

Some slides are taken from David Kauchak