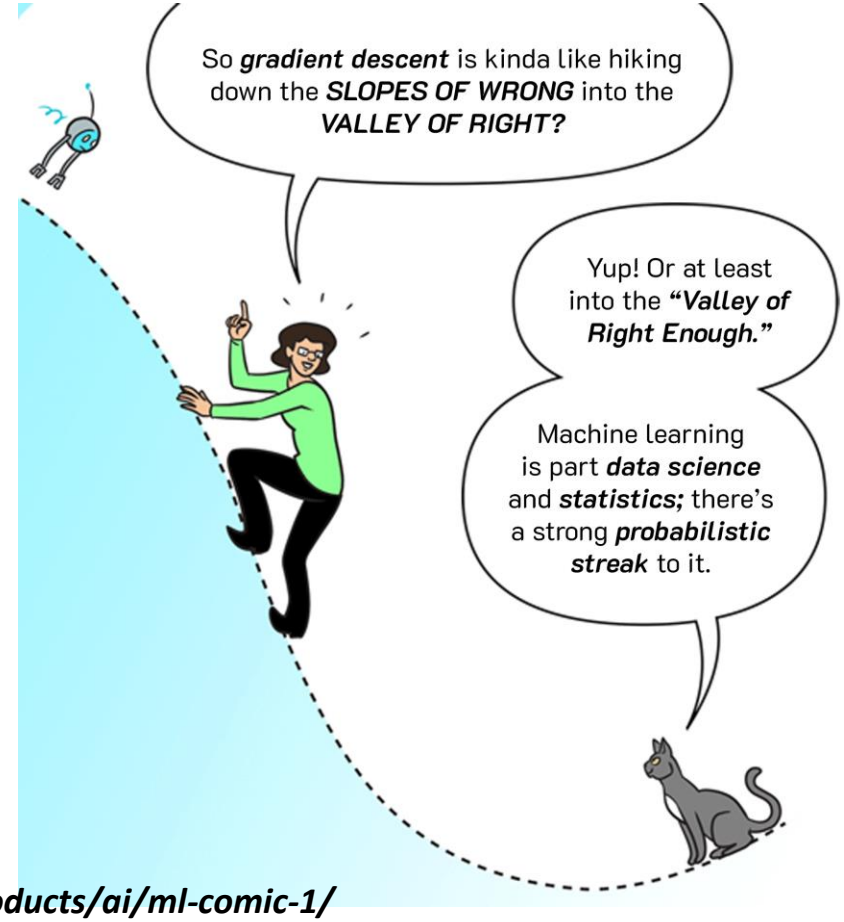
# Regularization




***https://cloud.google.com/products/ai/ml-comic-1/***

# How many have you heard of?

(Ordinary) Least squares

Ridge regression

Lasso regression

Elastic regression

Logistic regression

# 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)

$$\sum_{i=1}^{n} \mathbf{1}(y_i(w \cdot x_i + b) \leq 0)$$

3. develop a learning algorithm

$$\text{argmin}_{w,b} \sum_{i=1}^{n} \mathbf{1}(y_i(w \cdot x_i + b) \leq 0)$$

Find w and b that minimize the 0/1 loss

# 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

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

Find w and b that minimize the surrogate loss

# Surrogate loss functions

0/1 loss: $l(y, y') = 1[yy' \pounds 0]$

Hinge: $l(y, y') = \max(0, 1 - yy')$

Exponential: $l(y, y') = \exp(-yy')$

Squared loss: $l(y, y') = (y - y')^2$

# Finding the minimum

You're blindfolded, but you can see out of the bottom of the blindfold to the ground right by your feet. I drop you off somewhere and tell you that you're in a convex shaped valley and escape is at the bottom/minimum. How do you get out?

# Gradient Descent: Summary

1. Initialize the parameters w to some guess (usually all zeros, or random values)

2. Update the parameters (for each weight $w_j$):

$$w_j = w_j - \eta \frac{d}{dw_j} Loss(w_j)$$

3. Repeat step 2 until $\| \frac{d}{dw_j} Loss(w) \| < \theta$ or until the maximum number of iterations is reached.

# Perceptron learning algorithm

repeat until convergence (or for some # of iterations):
  for each training example ($f_1$, $f_2$, …, $f_m$, label):

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

~~if *prediction * label* ≤ 0: // they don't agree~~
    for each $w_j$:
      $w_j = w_j + f_j$*label
    $b = b$ + label

Note: for gradient descent, we always update

---

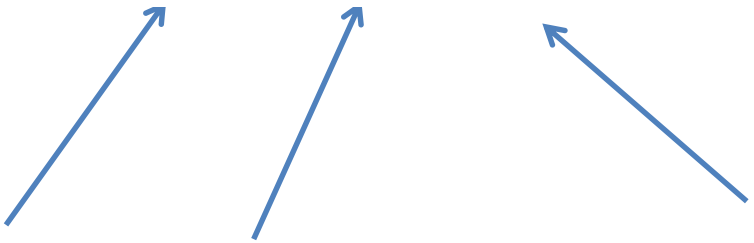
$$w_j = w_j + \eta y_i x_{ij} \exp(-y_i(w \cdot x_i + b))$$

or

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

# The constant

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



learning rate label prediction

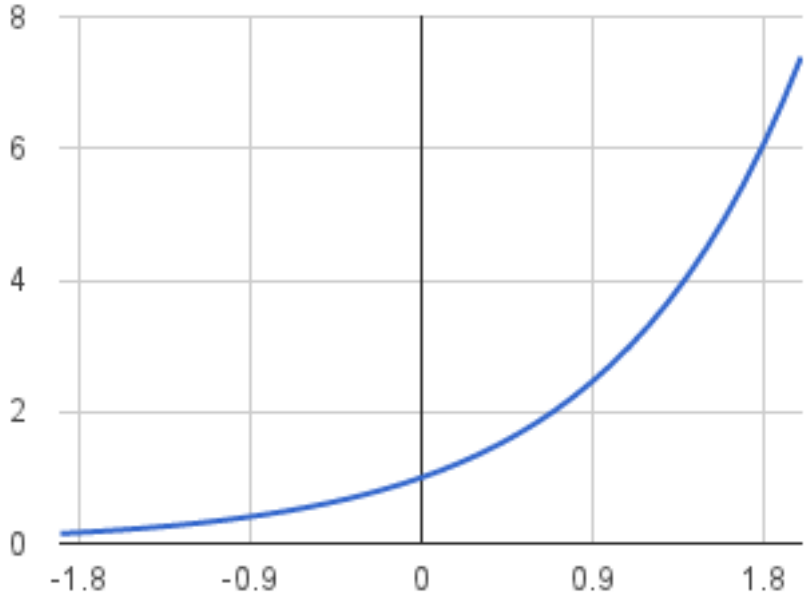
When is this large/small?

# The constant

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

label

prediction

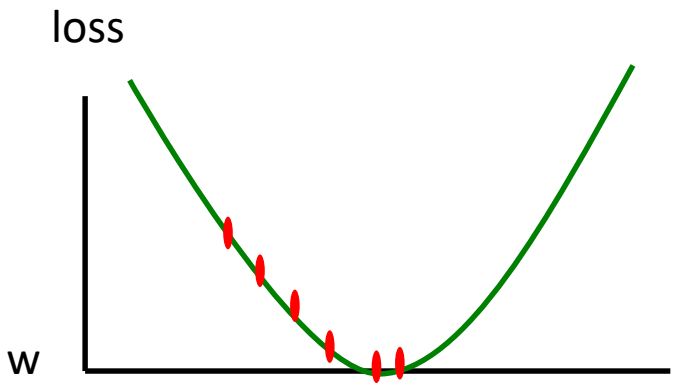



If they're the same sign, as the predicted gets larger there update gets smaller

If they're different, the more different they are, the bigger the update

# One concern

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




We're calculating this on the **training set**

We still need to be careful about overfitting!

The min w,b on the training set is generally NOT the min for the test set

How did we deal with this for the perceptron algorithm?

# Overfitting revisited: regularization

- A regularizer is an additional criterion 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 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

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

Should we allow all possible weights?

Any preferences?

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

# Regularizers

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

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

# Regularizers

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

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

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

# Common regularizers

sum of the weights

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

sum of the squared weights

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

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)

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

---

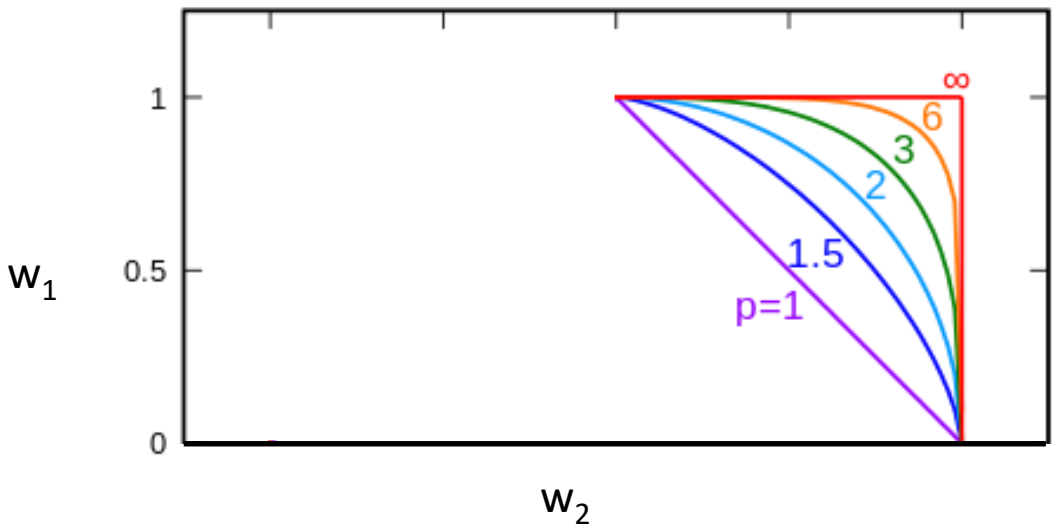
p-norm

$$r(w,b) = \left( \sum_j |w_j|^p \right)^{\frac{1}{p}}$$

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

Larger values of p discourage large weights more

# p-norms visualized



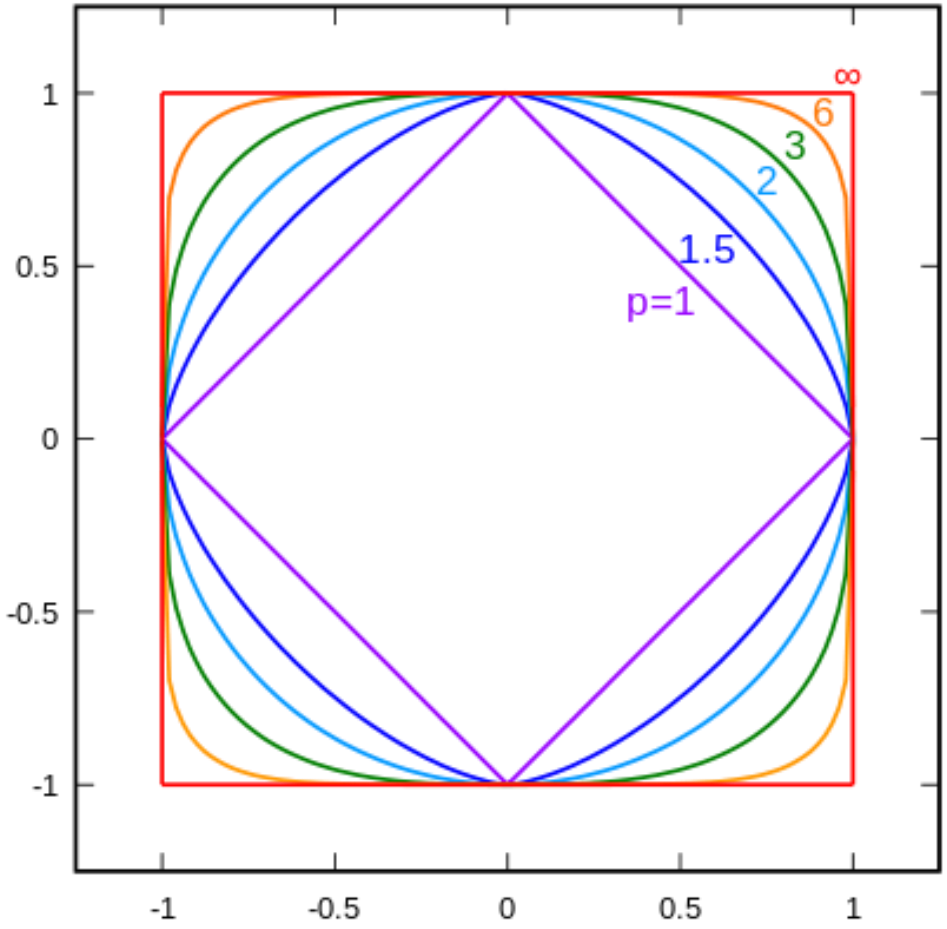

lines indicate penalty = 1

For example, if $w_1 = 0.5$

| p | $w_2$ |
|---|---|
| 1 | 0.5 |
| 1.5 | 0.75 |
| 2 | 0.87 |
| 3 | 0.95 |
| ∞ | 1 |

- **L1 norm (p = 1)** → promotes sparsity (drives some weights to zero).
- **L2 norm (p = 2)** → penalizes large weights but not too aggressively.
- **Smaller $p$ (p < 2)** → stronger sparsity and a sharper focus on small values.
- **Larger $p$ (p > 2)** → discourages large weights more strongly without promoting sparsity.

# 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

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

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

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

$$\text{loss}(y, y') = \sum_{i=1}^{n} \text{loss}(y_i, y_i')$$

3. develop a learning algorithm

$$\arg\min_{w,b} \left(\text{loss}(y, y') + \lambda \cdot \text{regularizer}(w)\right)$$

Find w and b that minimize

# Minimizing with a regularizer

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

$$\arg\min_{w,b} \left( \sum_{i=1}^{n} \text{loss}(y_i, y_i') \right)$$

- $y_i$ is the true value for the $i$-th data point,
- $y_i'$ is the predicted value for the $i$-th data point,
- $n$ is the number of data points in your dataset,
- $\text{loss}(y_i, y_i')$ is the loss function that quantifies the difference between the true label $y_i$ and the predicted label $y_i'$.

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

$$\arg\min_{w,b} \left( \sum_{i=1}^{n} \text{loss}(y_i, y_i') + \lambda \cdot \text{regularizer}(w) \right)$$

make convex

# Minimizing with a regularizer

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

$$\arg\min_{w,b} \left( \sum_{i=1}^{n} \text{loss}(y_i, y_i') \right)$$

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

$$\arg\min_{w,b} \left( \sum_{i=1}^{n} \text{loss}(y_i, y_i') + \lambda \cdot \text{regularizer}(w) \right)$$

convex as long as both loss and regularizer are convex

# p-norms are convex

$$r(w,b) = \left( \sum_j |w_j|^p \right)^{\frac{1}{p}}$$

p-norms are convex for p >= 1

# 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)

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

3. develop a learning algorithm

$$\arg\min_{w,b}\left(\sum_{i=1}^{n} \exp\left(-y_i\left(w^T x_i + b\right)\right) + \frac{\lambda}{2}\|w\|^2\right)$$

Find w and b that minimize

# Our optimization criterion

$$\arg\min_{w,b} \left( \sum_{i=1}^{n} \exp\left(-y_i \left(w^T x_i + b\right)\right) + \frac{\lambda}{2}\|w\|^2 \right)$$

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)

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

---

$$\arg\min_{w,b}\left(\sum_{i=1}^{n}\exp\left(-y_i\left(w^T x_i + b\right)\right) + \frac{\lambda}{2}\|w\|^2\right)$$

# Some more maths

**Steps to develop the learning algorithm:**

1. **Compute the Gradient**: To minimize the objective function, you need to compute the gradients of the loss function with respect to the parameters $w$ and $b$.
   - For the loss term $\exp(-y_i(w^T x_i + b))$, the gradient with respect to $w$ and $b$ will involve the chain rule of differentiation.

$$\frac{\partial}{\partial w}\left(\exp\left(-y_i(w^T x_i + b)\right)\right) = -y_i x_i \exp\left(-y_i(w^T x_i + b)\right)$$

$$\frac{\partial}{\partial b}\left(\exp\left(-y_i(w^T x_i + b)\right)\right) = -y_i \exp\left(-y_i(w^T x_i + b)\right)$$

2. **Include the Regularization**: The gradient of the regularization term $\frac{\lambda}{2}\|w\|^2$ is straightforward:

$$\frac{\partial}{\partial w}\left(\frac{\lambda}{2}\|w\|^2\right) = \lambda w$$

# 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)

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

---

$$w \leftarrow w - \eta \left( \sum_{i=1}^{n} -y_i x_i \exp\left(-y_i(w^T x_i + b)\right) + \lambda w \right)$$

$$b \leftarrow b - \eta \left( \sum_{i=1}^{n} -y_i \exp\left(-y_i(w^T x_i + b)\right) \right)$$

# The update

$$w \leftarrow w - \eta \left( \sum_{i=1}^{n} -y_i x_i \exp\left(-y_i(w^T x_i + b)\right) + \lambda w \right)$$

learning rate

direction to update

constant: how far from wrong

regularization

What effect does the regularizer have?

# The update

$$\arg\min_{w,b}\left(\sum_{i=1}^{n}\exp\left(-y_i\left(w^T x_i+b\right)\right)+\frac{\lambda}{2}\|w\|^2\right)$$

learning rate

direction to update

constant: how far from wrong

regularization

If $w_j$ is positive, reduces $w_j$
If $w_j$ is negative, increases $w_j$

} moves $w_j$ towards 0

# L1 regularization

$$\arg\min_{w,b}\left(\sum_{i=1}^{n}\exp\left(-y_i\left(w^T x_i+b\right)\right)+\lambda\sum_j|w_j|\right)$$

---

The gradient of this loss term with respect to $w$ and $b$ is:

- For $w$:

$$\frac{\partial}{\partial w}\left(\exp\left(-y_i(w^T x_i+b)\right)\right)=-y_i x_i\exp\left(-y_i(w^T x_i+b)\right)$$

- For $b$:

$$\frac{\partial}{\partial b}\left(\exp\left(-y_i(w^T x_i+b)\right)\right)=-y_i\exp\left(-y_i(w^T x_i+b)\right)$$

The L1 regularization term is:

$$\lambda\sum_j|w_j|$$

The gradient of $|w_j|$ with respect to $w_j$ is:

$$\frac{\partial}{\partial w_j}|w_j|=\operatorname{sign}(w_j)$$

So, the gradient of the L1 regularization term with respect to $w$ is:

$$\frac{\partial}{\partial w_j}\left(\lambda|w_j|\right)=\lambda\operatorname{sign}(w_j)$$

$$\frac{\partial}{\partial w_j}\left(\sum_{i=1}^{n}\exp\left(-y_i(w^T x_i+b)\right)+\lambda|w_j|\right)=-\sum_{i=1}^{n}y_i x_{ij}\exp\left(-y_i(w^T x_i+b)\right)+\lambda\operatorname{sign}(w_j)$$

# L1 regularization

The final update rule for $w_j$ is:

$$w_j \leftarrow w_j + h \left( \sum_{i=1}^{n} y_i x_{ij} \exp\left(-y_i(w^T x_i + b)\right) - \lambda \operatorname{sign}(w_j) \right)$$

Where:

- $h$ is the **learning rate**,
- $x_{ij}$ is the $j$-th component of the feature vector $x_i$ for the $i$-th data point.

What effect does the regularizer have?

# L1 regularization

$$w_j \leftarrow w_j + h\left(\sum_{i=1}^{n} y_i x_{ij} \exp\left(-y_i(w^T x_i + b)\right) - \lambda \operatorname{sign}(w_j)\right)$$

learning rate

direction to update

constant: how far from wrong

regularization

If $w_j$ is positive, reduces by a constant
If $w_j$ is negative, increases by a constant

} moves $w_j$ towards 0 ***regardless of magnitude***

# Regularization with p-norms

**L1:**

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

**L2:**

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

**Lp:**

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

How do higher order norms affect the weights?

# Model-based machine learning

develop a learning algorithm

$$\arg\min_{w,b}\left(\sum_{i=1}^{n}\exp\left(-y_i\left(w^T x_i+b\right)\right)+\frac{\lambda}{2}\|w\|^2\right)$$

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

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

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

---

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

# Many tools support these different combinations

Look at scikit learning package:

http://scikit-learn.org/stable/modules/sgd.html

# Common names

(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