LINEAR REGRESSION

Closed Form Equation

↓

A direct mathematical formula for computing best fit parameters

↓

It minimize the cost fun^{c} (MSE)

↓ by

Solving a matrix equation

Gradient Decent

↓

An iterative optimization algorithm that starts with random parameters and gradually improves them

↓

It minimize the cost fun^{c} (MSE)

↓ by

Calculating the gradient of cost fun^{c}

↓

Update parameters in opp dir

NORMAL EQN

Linear Regression Equation:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots \theta_n x_n$$

$\hat{y}$ = predicted Value

n = number of features

x_1 = ith feature

θ_j = j^{th} model parameter

θ_0 = bias term / intercept term

∴ Linear Regression more specifically predicts weighted sum of inputs

$$\hat{y} = h_\theta(x) = \theta \cdot x = \theta^T x$$

Vectorized form of Linear Regression

θ : model's parameter vector
having bias term θ_0 and feature weights θ_1 to θ_n

x : instance's feature vector
containing x_0 to x_n , with $x_0 = 1$.

h_θ : hypothesis function

To find parameters we will minimize RMSE or say MSE as it leads to same result

$$MSE(X, h_\theta) = \frac{1}{m} \sum_{i=1}^{m} (\theta^T x^{(i)} - y^{(i)})^2$$

to find the value of θ which minimize the cost function

Normal Equation

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

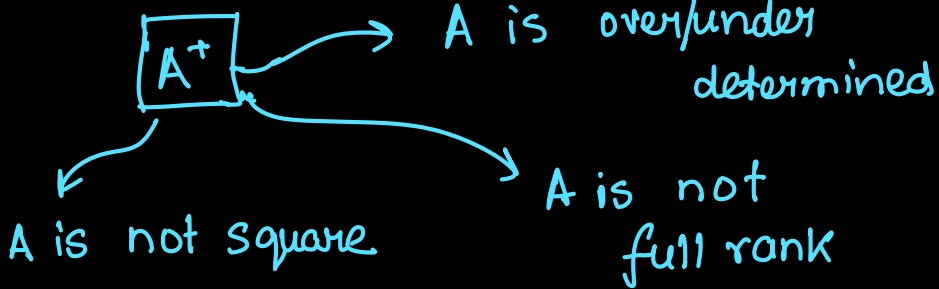
$\hat{\theta}$ value that minimize the cost fun^c

y is the vector of target values containing $y^{(1)}$ to $y^{(m)}$

Implementation of Linear Reg using Normal Eqn:

```python
import numpy as np

# Sample Data
# X: Feature matrix (m x n), y: Target vector (m x 1)
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([2, 4, 6, 8, 10])

# Step 1: Add a bias (intercept) term -> add a column of ones to X
m = X.shape[0]
X_b = np.c_[np.ones((m, 1)), X]  # Shape: (m x n+1)

# Step 2: Apply the Normal Equation
# θ = (Xᵀ X)^(-1) Xᵀ y
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)

print("Parameters (θ):", theta_best)

# Step 3: Predict for new data
X_new = np.array([[0], [6]])
X_new_b = np.c_[np.ones((X_new.shape[0], 1)), X_new]  # add bias
y_predict = X_new_b.dot(theta_best)

print("Predictions:", y_predict)
```

Pseudo Inverse or Moore-Penrose Inverse:



$$\theta = (X^T X)^{-1} X . y$$

$$= X^+ y$$

How computed ??

Singular Value Decomposition (SVD):

$$A = U \Sigma V^T \Rightarrow A^+ = V \Sigma^+ U^T$$

- Gives best appoximate solution

$$X = U \Sigma V^T$$

Transpose of orthogonal matrix

Orthogonal matrix (left singular vector)

diagonal matrix of singular values

$$X^\dagger = V \Sigma^\dagger U^T$$

To compute $\Sigma^\dagger$:

1. Take Σ - the diagonal matrix of singular values
2. Set to zero any singular value smaller than a small threshold (to avoid dividing by near zero)
3. Take the reciprocal ($1/\sigma_i$) for each non zero value
4. Transpose the result

Complexity in Computation:

Normal Equation computes

$X^T X$ which is a

$(n+1) \times (n+1)$ matrix

$O(n^{2.4})$ to $O(n^3)$

If no. of features double :

5.3 to 8 times more time

But, SVD approach takes $O(n^2)$

∴ Normal or SVD approch is long enough to train but fast enough to predict...

For number of instance m :

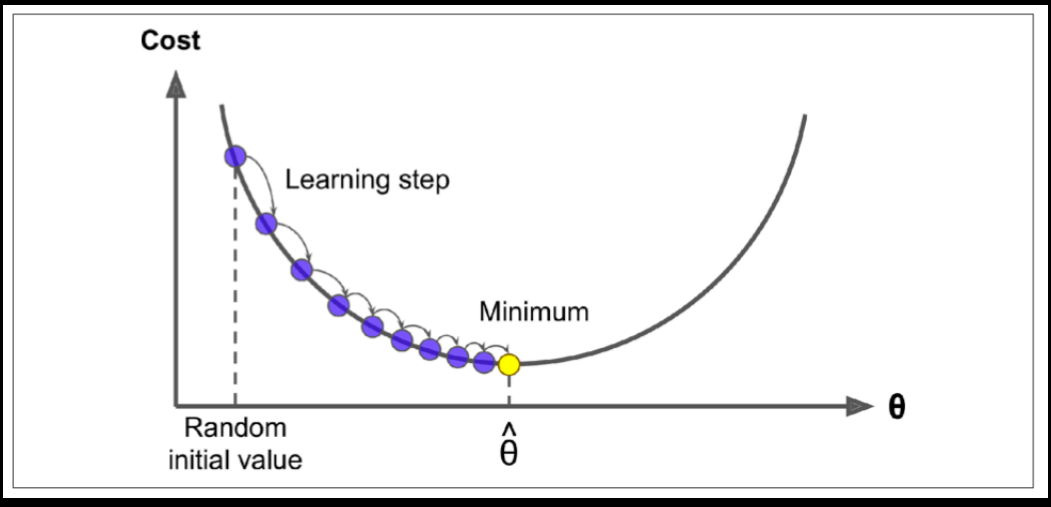
$O(m) \rightarrow$ linear

If (instance x2) : (prediction time x2)

GRADIENT DESCENT

generic optimization algorithm

↳ tweak parameters iteratively in order to minimize cf



Start by feeding θ with random values

called random initialization

↓

Take one step at a time each attempted to decrease the cost function

↓

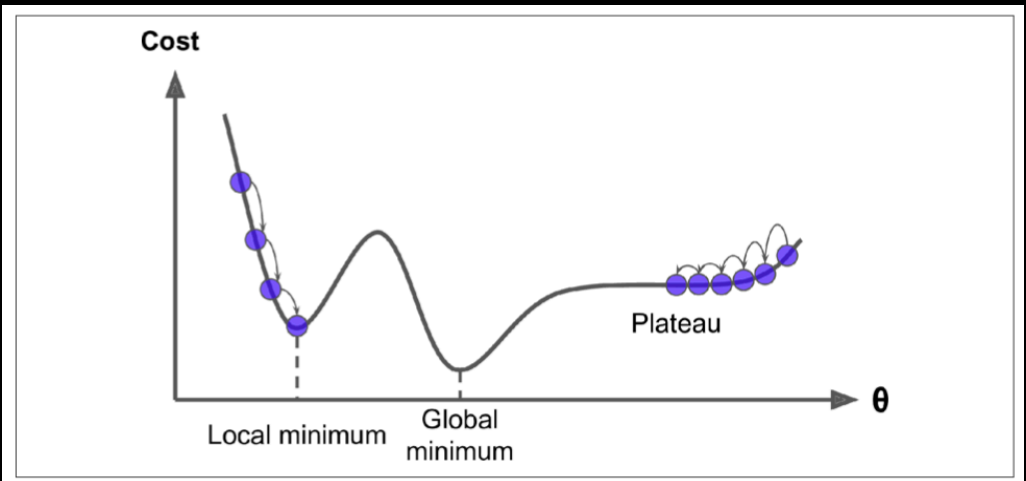
Untill the algorithm converges to minimum

Size of step → determined by learning rate hyperparameter

→ not too small and not too large

Problem with GD:

- Local minima
- Plateau terrain
- Cut off



But fortunately the MSE cost function is a convex fun^{c}:

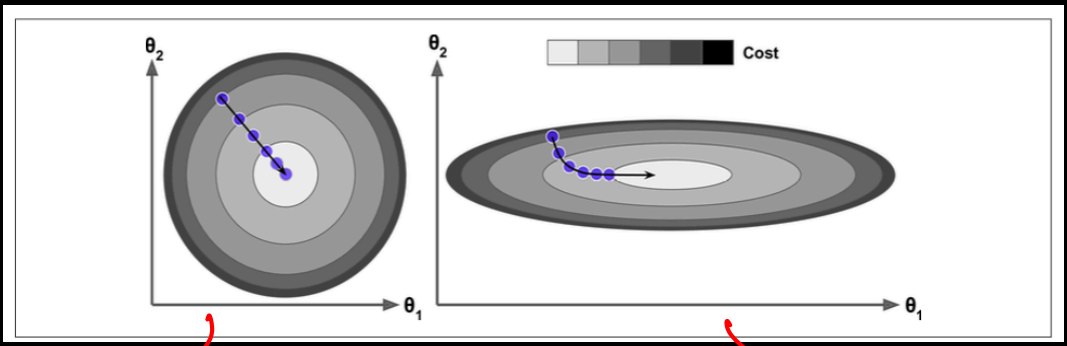
- It will only have one minima and it will be global minima
- It will be smooth

(Convexity + Smoothness)

The cost function has a shape of bowl

can be elongated bowl if the features have different scale

** Before using GD we should ensure all features have same scale



will take smaller time as it is scaled

will take longer time as features are not scaled

More parameters the mode have

↓

more dimensions this space has

↓

more hard to search

BATCH GD

$$MSE(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta^T x^{(i)} - y^{(i)} \right)^2$$

$$\frac{\partial}{\partial \theta_j} MSE(\theta) = \frac{2}{m} \sum_{i=1}^{m} \left(\theta^T x^{(i)} - y^{(i)} \right) \cdot x_j^{(i)}$$

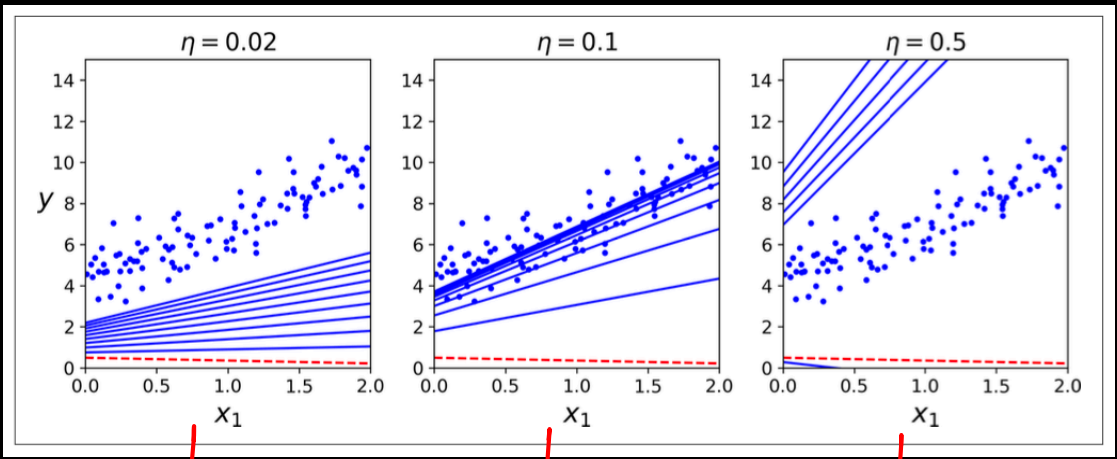
gradient vector of the cost fun^c:

$$\nabla_\theta MSE(\theta) = \begin{bmatrix} \frac{\partial}{\partial \theta_0} MSE(\theta) \\ \frac{\partial}{\partial \theta_1} MSE(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} MSE(\theta) \end{bmatrix} = \frac{2}{m} X^T (X\theta - y)$$

$$\theta^{(\text{next step})} = \theta - \eta \nabla_\theta MSE(\theta)$$

— gradient decent step

η = learning rate



too slow

fit

too fast

How to set number of iteration:

A simple solution is set the no. of iteration to large number.

How to know when to stop:

when norm of gradient vector becomes smaller than a tiny number ε (the tolerance)

$$\|\nabla_\theta MSE(\theta)\| < \varepsilon \ (1 \times 10^{-6})$$

Convergence Rate:

For a batch GD the convergence rate is

$$O\left(\frac{1}{\varepsilon}\right)$$

To get within optimal solution, it may take $\frac{1}{\varepsilon}$ iteration

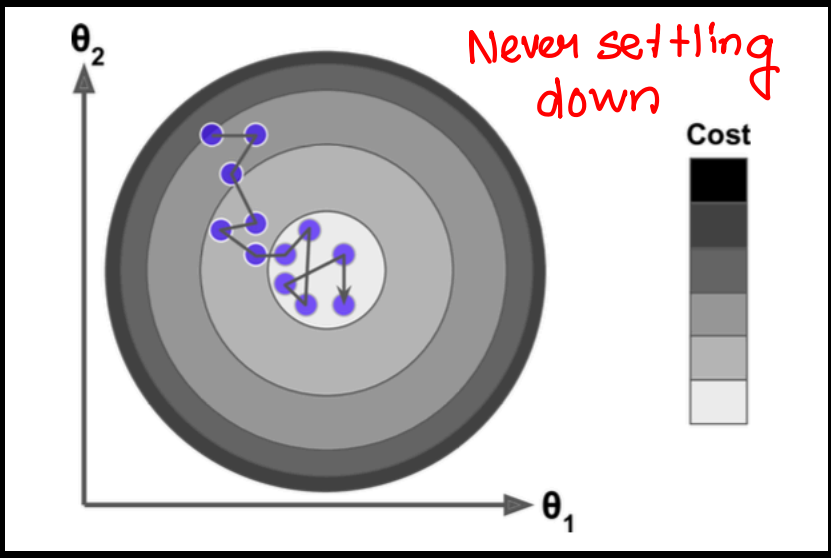
STOCHASTIC GD

- The batch gd uses whole training set and computes gradient at each step
- But stochastic gd uses random instances
- Hence it is faster to compute
- The algoritm is less regular than batch gradient

↓

Untill gently reaching minima it will bounce up and down

- Once the algorithm stops, the final parameter values are good but not optimal.



- This randomness less to jump out of local minimas but it means the algorithm can never settle down.
- One solution is reducing the learning rate.

The steps start out large but then gets smaller and smaller

This process is known as simulated annealing

- The function that determines the learning rate at each iteration is called the learning schedule.

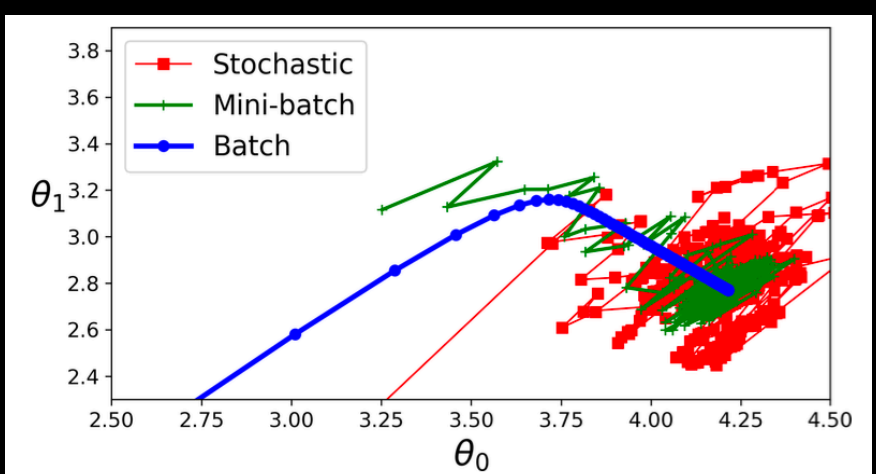
- While using SGD the training instances should be distributed identically and independently (IID).

- If the instances are sorted by label, the SGD will optimize for one label and then for another

- SGD **depends heavily on data shuffling** to make unbiased parameter updates.
- Without shuffling, **batch gradients become highly biased**, especially early in training.
- This is especially dangerous for **imbalanced datasets** or **online learning**.

MINI BATCH GD

- Mini Batch GD computes gradients on small random sets of instances called mini-batches.
- One advantage of MBGD is that you get a performance boost over SGD.
- It may be harder for it to escape local minimas
-



⇒ Batch one take more time but stops once reaching the global minima

⇒ Mini Batch and stochastic continue to walk around

Table 4-1. Comparison of algorithms for Linear Regression

Algorithm	Large *m*	Out-of-core support	Large *n*	Hyperparams	Scaling required	Scikit-Learn
Normal Equation	Fast	No	Slow	0	No	N/A
SVD	Fast	No	Slow	0	No	`LinearRegression`
Batch GD	Slow	No	Fast	2	Yes	`SGDRegressor`
Stochastic GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`
Mini-batch GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`

POLYNOMIAL REGRESSION

Polynomial Features all number of features upto a given degree.

Ex:- If there are two features:

a and b

for Polynomial degree = 3

a^3, b^3, b^2, a^2

and also, ab, a^2b, ab^2

*** For PolynomialFeature (degree = d) and n feature:

$\frac{(n+d)!}{d!\,n!}$ feature array.

It basically creates $\frac{(n+d)!}{d!\,n!}$ features and do linear regression between them.

LEARNING CURVES

How to know which curve to apply?

- If training set error low, validation set error high } underfitting
- If both high } overfitting

- Another way is to look at learning curves:
  Train the model several times on different sized subsets of the training set

* If model underfitting: No use of adding more training examples
  If model overfitting: Feed it more training data untill Val error reaches train error

BIAS-VARIANCE

Model's generalization Error:

Bias + Variance + Irreducible Error

- Bias
  - due to wrong assumptions
  - (Ex: Assume data is linear but actually quadratic)
  - A high bias model is most likely to overfit
- Variance
  - due to excessive sensitivity to small variations in training data
  - A model with high degree of freedom is likely to have high variance leading to overfitting
- Irreducible Error
  - due to noisiness of data
  - only way to reduce this is to cleanup the data

* If model is complex:
  - low bias
  - high variance

Trade off

If model is simple:
  - low variance
  - high bias

RIDGE REGRESSION

- A good way to reduce overfitting is to regularize the model.
  - constrain it
  - lower degree of freedom

- Tikhonov Regularization is the regularized version of Linear Reg

- A regularized term: $\alpha \sum_{i=1}^{n} \theta_i^2$ is added to CF
  - only added during training
    - once model is trained
      - use unregularized paramater

* CF used during training will not always be used for testing:

Reason: good training cost function should have optimization friendly derivative but the testing performance measure should be as close as possible to final objective

Ridge Regression cost function

$$J(\theta) = MSE(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

this α hyperparameter controls how we want to regularize

if $\alpha = 0$;
Rigde Reg ≈ Lin Reg

α = very large
Model goes through avg (simpler model)

If we define w as a vector of feature weights (θ_1 to θn):

then the regularized term is to

$$\frac{1}{2}(\|w\|_2)^2$$

for gradient decent add αw

* Imp to Scale before apply Rigde Rig

α Value	Effect on Model	Bias	Variance
Very Low	Overfit	Low	High
Medium	Balance fit	—	—
Very High	underfit	High	Low

Ridge Regression Closed Form Soln:

$$\hat{\theta} = (X^T X + \alpha A)^{-1} X^T y$$

With Ridge: the model is forced to keep coefficients small, leading to smaller curve

↓

without it high degree polynomial wiggle a lot (overfit)

* Instead of shrinking coefficients the Ridge Rig can also constrain weights

And solve Constrained Optimization Problem

Minimize $\sum_{i=1}^{m} (y^i - x^i)^2$ subject to $\sum_{i=1}^{n} \theta_j^2 \leq t$ ← constrain

* Bayesian Interpretation of Ridge:

$$p(\theta) \propto \exp\left(-\frac{\lambda}{2}\|\theta\|_2^2\right)$$

LASSO REGRESSION

↳ Least Absolute Shrinkage and Selection Operator Regression

- uses l_1 norm just like Ridge uses l_2

Lasso Reg Cost Function:

$$J(\theta) = MSE\theta + \alpha \sum_{i=1}^{n} |\theta_j|$$

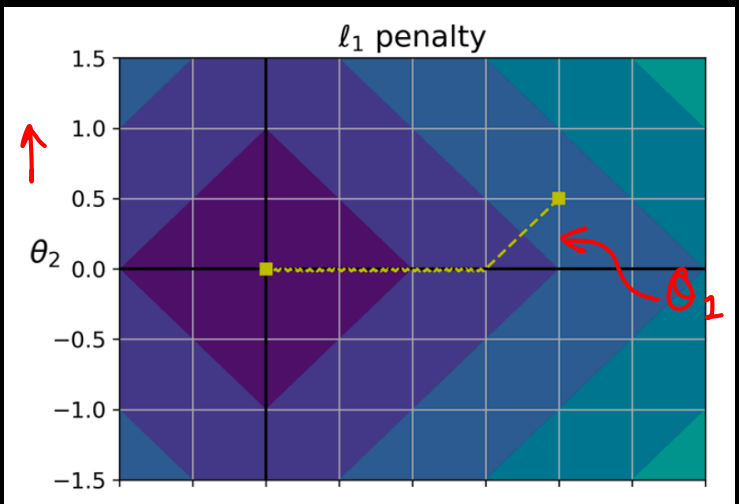
- Lasso Reg tends to eliminate the weights of least important features (set → zero)

↓

Or one can say Lasso Reg automatically performs feature selection and outputs a sparse Model

Lasso Vs Ridge Regularization:

Ex: For $\theta_1 = 2$ and $\theta_2 = 0.5$

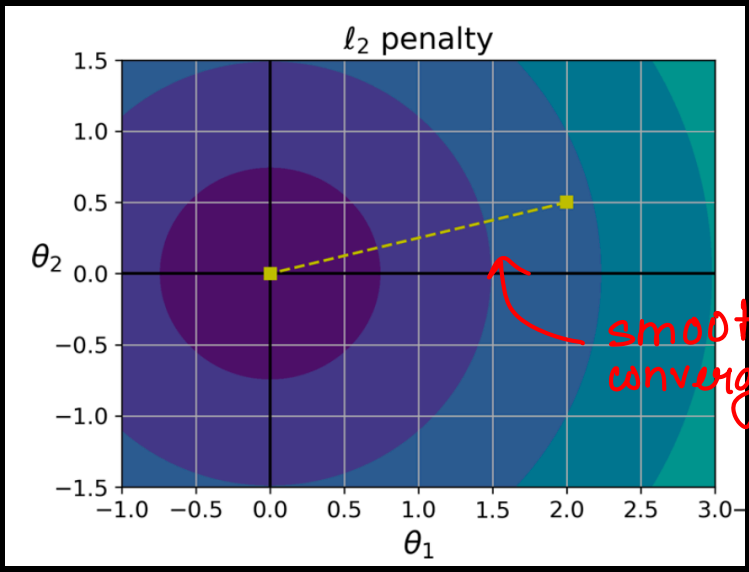


Loss func: $\ell_1 = |\theta_1| + |\theta_2|$

Diamond shaped counter (Manhattan norm)

$$\text{gradient} = \frac{d}{d\theta_j}|\theta_j| = \begin{cases} 1 & \theta_j > 0 \\ -1 & \theta_j < 0 \\ \text{undef} & \theta_j = 0 \end{cases}$$

∴ gradient decent moves in straight path based on gradient direction



Loss fun^{c} :

$$l_2 = \theta_1^2 + \theta_2^2$$

circular contours

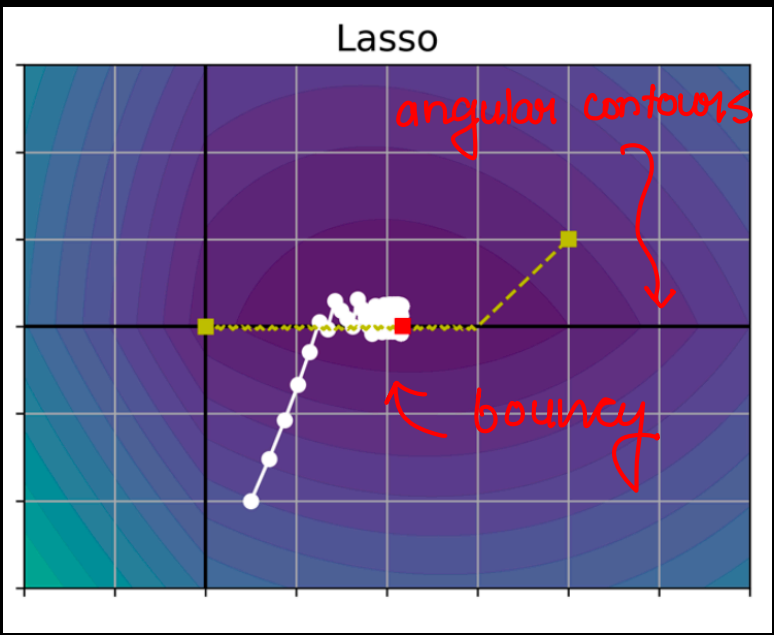
Euclidean Norm

$$\text{gradient} = \frac{d}{d\theta_j}\theta_j^2 = 2\theta_j$$

Smoothly decays to zero as $\theta_j \to 0$

Feature	L1 (Lasso)	L2 (Ridge)
Derivative	Constant ±1 or subgradient at 0	Smooth: $2\theta_j$
Behavior near zero	Sharp, aggressive pull toward 0	Gentle, smooth shrinkage
Effect	Sets some weights **exactly to zero**	Keeps all weights, **shrinks all**

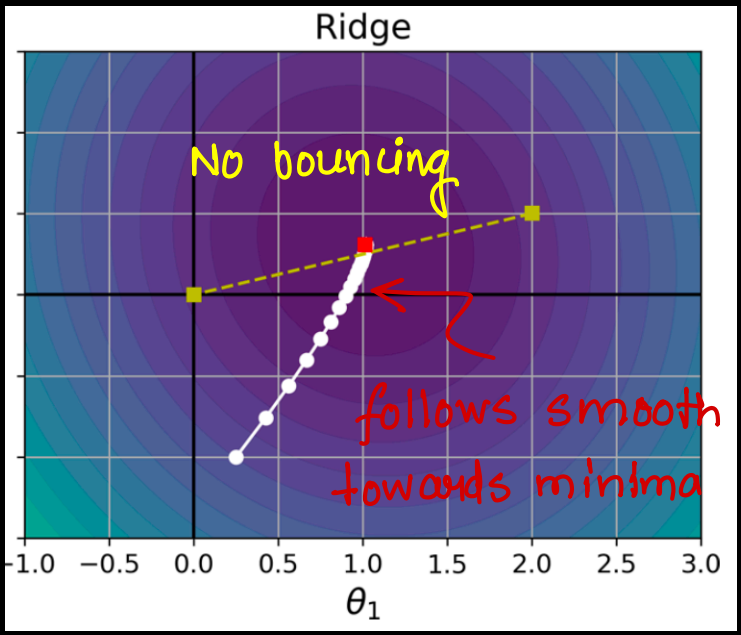
•) Cost fun^c with Lasso Reg fun^c (MSE + ℓ_1)
↳ (actually Smooth bowl)



•) Analogy ball rolls into a corner then along edges
Constant bounces at corners

•) Cost func^c with Ridge Reg func^c:

(MSE + l_2)



•) Smooth convergence

Property	Lasso (L1)	Ridge (L2)
Penalty Shape	Diamond (Manhattan norm)	Circle (Euclidean norm)
Encourages Sparsity?	✅ Yes, zeroes out weights	❌ No, shrinks but keeps all
Feature Selection?	✅ Yes	❌ No
Gradient Behavior	Constant, bounces at corners	Smooth, decays near optimum
Path to Optimum	Axes-aligned, piecewise	Smooth, radial inward
Good for High-Dim?	✅ When only few features matter	✅ When all features contribute

Lasso-Regression Subgradient Vector

$$g(\theta, J) = \nabla_\theta MSE(\theta) + \alpha \begin{pmatrix} sgn(\theta_1) \\ sgn(\theta_2) \\ \vdots \\ sgn(\theta_n) \end{pmatrix}$$

$$sgn(\theta_i) = \begin{cases} -1 & \theta_i < 0 \\ 0 & \theta_i = 0 \\ +1 & \theta_i > 0 \end{cases}$$

ELASTIC NET

- Middle ground btw Ridge and Lasso:
- We can control the mix ratio:

  when $r=0 \Rightarrow$ Ridge Regression

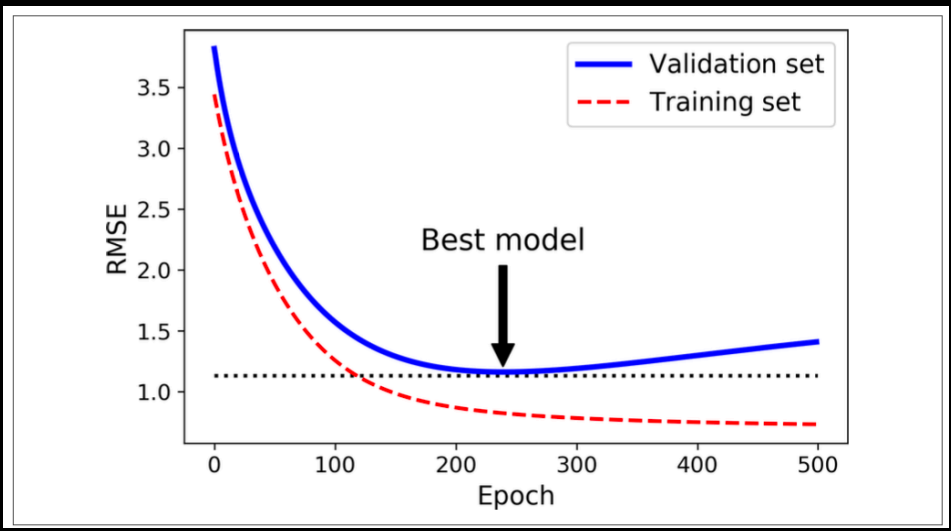
  $r=1 \Rightarrow$ Lasso Regression

Elastic Net Cost Function:

$$J(\theta) = MSE(\theta) + r \alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2} \alpha \sum_{i=1}^{n} \theta_i^2$$

Lasso gives sparsity
Ridge gives stability
} Net give both

★ Early Stopping



- RMSE decreases for both training and validation

↓

But after sometime the model overfits

↓

Hence it is imp to stop when the model hit minimum RMSE

- warm_start = True
  allows incremental training across epochs

Clone()
Best model stored using Clone

LOGISTIC REGRESSION

- It is used to estimate the probability that an instance belongs to a particular class.
- If prob (>50%) it belongs to that class (+ve class) or it does not (-ve class)

Logistic Reg Model (Vectoized Form)

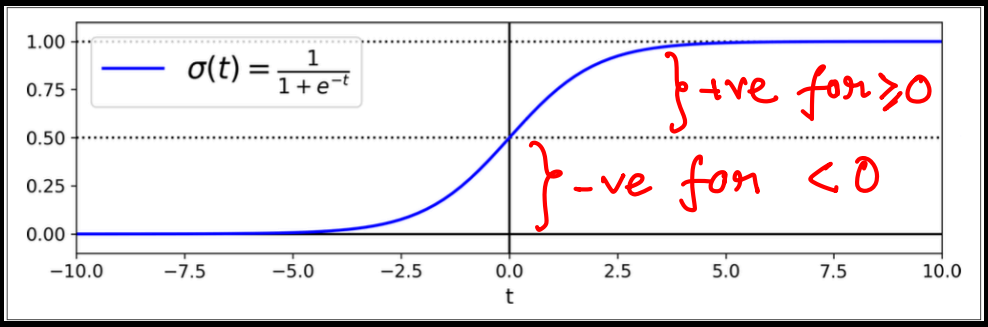
Estimated Prob:

$$\hat{p} = h_\theta(x) = \sigma(X^T\theta)$$

$\sigma(\cdot)$ = sigmoid func used to get value btw 0 to 1

Logistic Func:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$



•) Once the Log Reg estimated the prob

$$\hat{p} = h_\theta(x)$$

$x \in$ positive class

It predicts $\hat{y}$ easily

Log Reg Model Prediction:

$$\hat{y} = \begin{cases} 0, & \hat{p} < 0.5 \\ 1 & \hat{p} \geq 0.5 \end{cases}$$

So,

$\sigma(x^T\theta)$

value < 0

$\sigma(< 0)$

$= < 0.50$

$=$ -ve class

value ≥ 0

$\sigma(\geq 0)$

$= \leq 0.50$

$=$ +ve class

Score + is also known as logit

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

Also called log odds,

since log of ratio between

+ve class and -ve class

$$(t = X^T\theta)$$

$$p = \sigma(t) = \frac{1}{1+e^{-t}}$$

$$\Rightarrow \quad \frac{p}{1-p} = \frac{1+e^{-t}}{1-1-e^{-t}}$$

$$\Rightarrow \quad = \frac{1+e^{-t}}{e^{-t}}$$

$$\Rightarrow \quad = e^{t}$$

$$\Rightarrow \quad \log\left(\frac{p}{1-p}\right) = \log(e^{t})$$

$$\Rightarrow \quad \text{logit}(p) = t$$

Cost Fun^c of single training instance:

$$c(\theta) = \begin{cases} -\log(\hat{p}) & \text{if } y=1 \\ -\log(1-\hat{p}) & \text{if } y=0 \end{cases}$$

Logistic Regression Cost Fun^c:
(log loss)

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(\hat{p}^{(i)}) + (y^{(i)}\log(1-\hat{p}^{(i)})\right]$$

Log Cost Fun^c partial derivative:

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\sigma(\theta^T x^{(i)}) - y^{(i)})x_j^{(i)}$$

★ Decision Boundary:

- A surface that seperates different classes in a classification problem
- It represents a threshold
- A line in 2D
  A plane in 3D
  and hyperplane in higher dimension

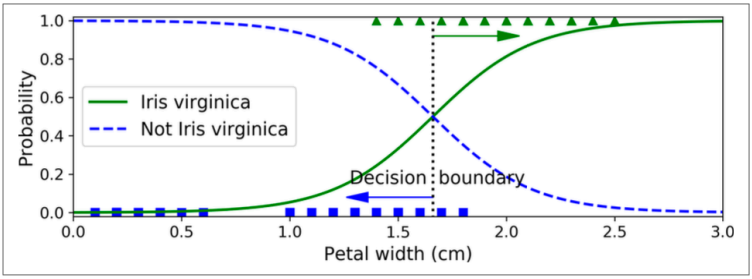
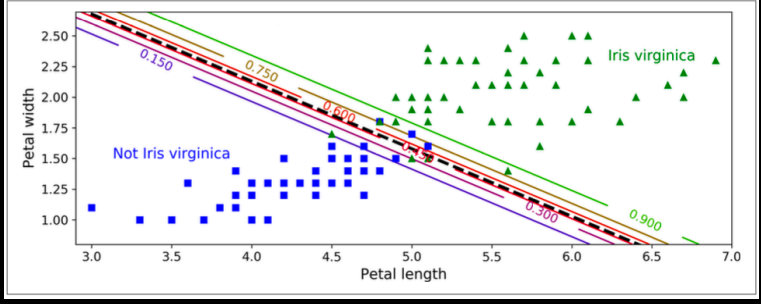


Figure 4-23. Estimated probabilities and decision boundary



[Linear Decision Boundary]

SOFTMAX REGRESSION

- also known as multinomial logistic Regression

Softmax Score for class k:

$$S_k(x) = x^T \theta^k$$

- Each class has its own dedicated parameter vector $\theta^{(k)}$

All these vectors are stored as rows in a parameter matrix

- When given instance x

↓

Softmax Regression first calculates score $s_k(x)$ for each class k

↓

then estimates the prob of each class by apply the softmax func (also called Normalized Exp)

↓

Once computed probability classifier predicts the class with highest estimated probability

SoftMax function:

$$\hat{p}_x = \sigma(s(x))_k = \frac{\exp(s_k(x))}{\sum_{j=1}^{K} \exp(s_j(x))}$$

K is number of class

$s(x)$ vector containing the scores

$\sigma(s(x))$ estimated prob that the instance x belongs to class k given the score of each instance

SoftMax Reg Classifier Prediction:

$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma(s(x))_k$$

$$= \underset{k}{\operatorname{argmax}}\ s_k(x)$$

$$= \underset{k}{\operatorname{argmax}}\ \left(\left(\theta^{(k)}\right)^T x\right)$$

- argmax operator returns the value of variable for which the function maximizes
- In this operator the value of k for which the estimated prob $\sigma(s(x))_k$ maximize

* SoftMax should only be used with mutually exclusive classes

It is not multilabel or multioutput

ENTROPY

- Measures the uncertainty in a prob dist.

Entropy for discrete dis P:

$$H(P) = -\sum_{x} P(x) \log P(x)$$

High Entropy = more uncertainty (uniform dist)

Low Entropy = less uncertainty (one class has prob 1)

Cross-Entropy:

$$H(P, Q) = -\sum P(x) \log Q(x)$$

P: True distribution

Q: Predicted Distribution

- Used as loss function in classification

KL Divergence:

$$D_{KL}(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

= Cross entropy - Entropy

- Not Symmetric

Cost Entropy Cost Function:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$

For $(K=2)$,

The equation becomes log loss

- goal is to minimize $J(\theta)$, cost function
  - typically done by gradient descent
- Used in
  - SoftMax Classifier
  - Binary Classifier