Chap4

Training Models

Written by Jin Kim

113~114 - 이번 챕터에서 할 것 소개

- Linear Regression model 을 트레인할것임.
  - 두 가지 방법으로 할것
    - “closed-form” equation 을 이용 -> 최상의 파라미터를 계산할수있음
    - iterative optimization(반복 최적화) -> GD(Gradient Descent)를 이용하여 점진적으로 모델 파라미터가 cost function을 최소화하게 만듦. 결과는 위 첫번째 방법과 같아짐
  - Batch GD, Mini-batch GD, Stochastic GD 등도 알아볼것임.
- Polynomial Regression을 알아볼것임.
  - Nonlinear datasets을 트레인 가능함.
  - 파라미터가 더 많기때문에, Overfit될 가능성이 있음.
- Classification에서 잘 사용되는 두가지 모델에 대해서도 알아볼것임.
  - Logistic Regression 과 Softmax Regression

114~115 Linear Regression

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

Y hat : 예측값

n : Feature의 갯수

xi : i번째 Feature의 값

세타 : 모델 파라미터

세타0 : 바이어스

115 Linear Regression 계속

$$\hat{y} = h_{\theta}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

와 같이도 쓸 수 있음.

x : feature vector

θ · x : dot product

-> $\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n.$

와 같음

h : hypothesis(가설/추정) function

θ 와 x 가 컬럼 벡터면 -> $\hat{y} = \boldsymbol{\theta}^{\mathrm{T}} \mathbf{x}$

와 같이 구해야함

115~116 어떻게 잘 트레인 하는가?
= 어떻게 파라메터를 설정하는가?

- RMSE(Root Mean Square Error)를 최소화하는 파라메터를 설정하는게 좋다.
  - 최소화할 수 있는 θ의 값
  - Mean Square Error (MSE) 를 최소화하는것도 결과가 같으며, 더욱 간단히 할 수 있음.
- MSE 공식 : $\mathrm{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)}\right)^2$

116 The Normal Equation

- 최소화하는 세타를 찾으려면, closed-form solution 이 있다.
  - = 정답을 찾을 수 있는 방정식이 있다.
  - 이것은 Normal Equation이라고 불림.

$$\widehat{\theta} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1} \quad \mathbf{X}^T \quad \mathbf{y}$$

- 세타햇 = 최소화된 세타값
- y = 타겟값(y(1) 부터 y(m)까지를 포함하고있음)
- Colab

119 pseudoinverse

- Pseudoinverse는 the Normal Equation 과 같이 세타햇을 계산할 수 있는 방정식이지만, Normal Equation이 사용될 수 없는 경우에도 사용 될 수 있음.
  - 예. $\mathbf{X}^T\mathbf{X}$ 가 뒤집힐 수 없는경우(Singular) 에는 Pseudoinverse 로 세타햇을 계산 가능함.
- np.linalg.pinv() 을 이용하여 간단히 계산 가능

119 Computational Complexity

- Normal Equation로부터 $\mathbf{X}^T\mathbf{X}$ 의 역행렬을 구할때 걸리는 시간은,
- $O(n^{2.4})$ to $O(n^3)$ 와 같다
- Feature의 갯수가 2배가 되면, $2^{2.4} = 5.3$ to $2^3 = 8$ 로 5.3에서 8만큼의 시간이 걸림

- Scikit-Learn의 LinearRegression인 SVD의 경우, $O(n^2)$ 와 같다.
- Feature의 갯수가 2배가 되면, 시간이 4배가 걸림

119 Normal Equation the SVD의 장/단점

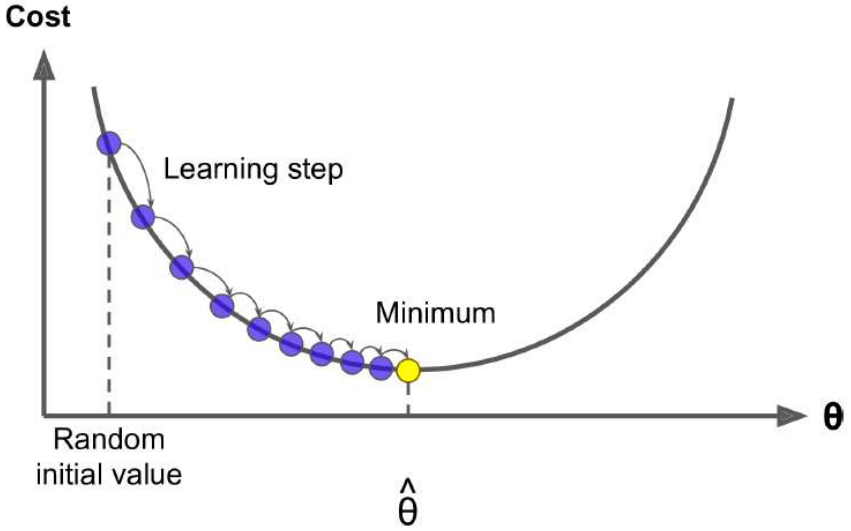
- 단점 : Feature의 갯수가 많을경우 시간이 많이 걸릴 수 있음
- 장점 : Instance의 수에 대해 선형이다 = 메모리에 들어 갈 수만 있으면, 큰 트레이닝 세트라도 효율적으로 다룰 수 있음. 한번 트레이닝 된 모델은 예측이 매우 빠름.

119 Computational Complexity

- Predict 를 할 때 소요되는 시간은 Instance의 수 + Feature의 수와 Linear관계임.
  - 예. Instance가 2배로 늘었다 -> 소요시간 2배

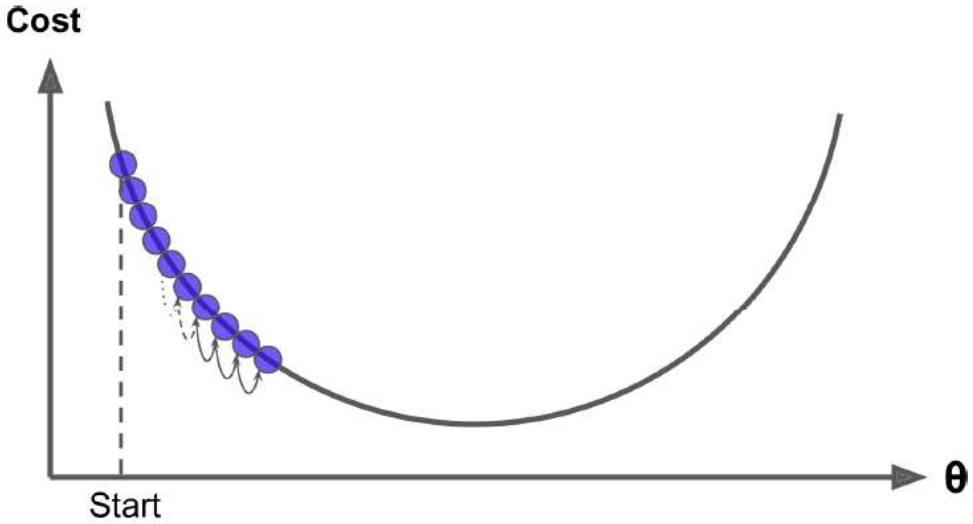
119~120 Gradient Descent

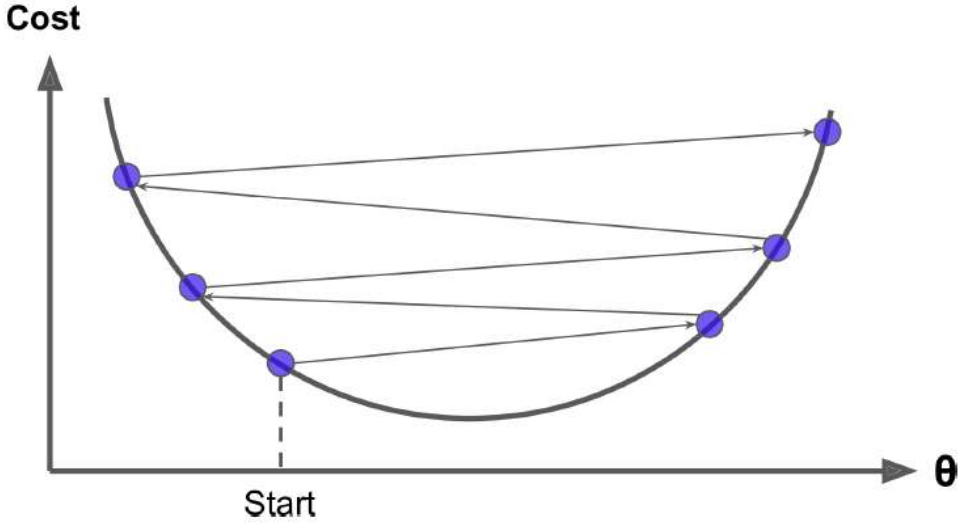
- local gradient of the error function(에러펑션의 경사)가 0이 되면 최소값을 얻는것임. 최초값은 랜덤하게 정할 수 있음



120 Gradient Descent

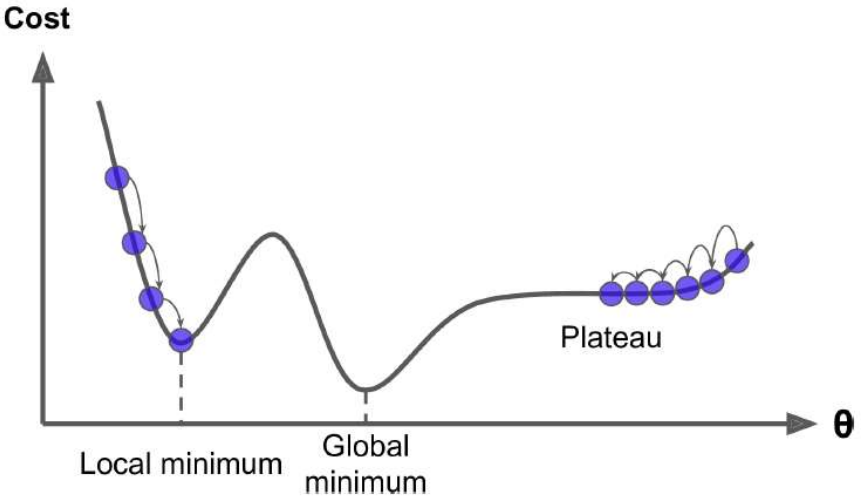
- 중요한 파라미터 : 스텝의 크기 = Learning rate
  - 너무 작을경우 : 시간이 많이 걸림
  - 너무 클 경우 : 왔다갔다 할 수 있음



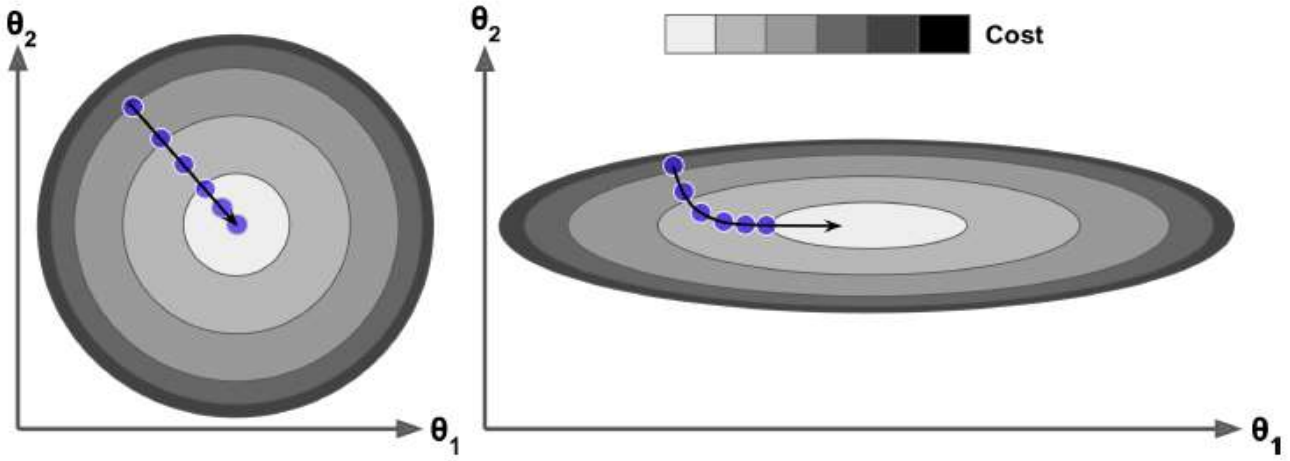


122 Gradient Descent(계속)

- Cost function이 위에서 봤던 그릇모양이 아닐 경우도 많음.
  - 이 경우에는 최소값을 구하기가 어려울 수 있음
- 또한, 최초의 세타값을 이상한 위치에 잡아 경사하강법을 진행 할 경우, 엉뚱한 값을 최소값으로 오인할 수 있음. 해결법 -> *convex function*(볼록함수)



122 convex function



- 두 지점을 고를 경우 위와 같은 형태의 그래프가 그려질 수 있음
- Gradient Descent는 이 특성을 이용해global minimum으로 확실하게 갈 수 있음
- 좌측그림과 우측그림처럼 cost function의 형태가 바뀔 수는 있음.

123 Gradient Descent사용시 주의사항

- 모든 Feature들이 비슷한 스케일에 있도록 해야함.
  - Ex. Scikit-Learn's StandardScaler
  - 이렇게 안하면 시간이 훨씬 많이 걸릴 수 있음.
- '모델을 트레이닝 한다' = '모델 파라미터들 최적의 조합을 통해 cost function의 최소값을 찾는다' 라고 생각 할 수도 있음.

123~124 Batch Gradient Descent

- 편미분(partial derivative) 를 계산하는방법.

Equation 4-5. Partial derivatives of the cost function

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

Equation 4-6. Gradient vector of the cost function

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

- 매 스텝마다 전체의 Batch를 사용하여 트레이닝 시킴
- 데이터가 클 경우 굉장히 느림
- 수십만가지의 데이터를 Linear Regression 모델로 트레인 시킬 경우, 경사하강법을 이용하면 Normal Equation 이나 SVD decomposition 보다 빠름.

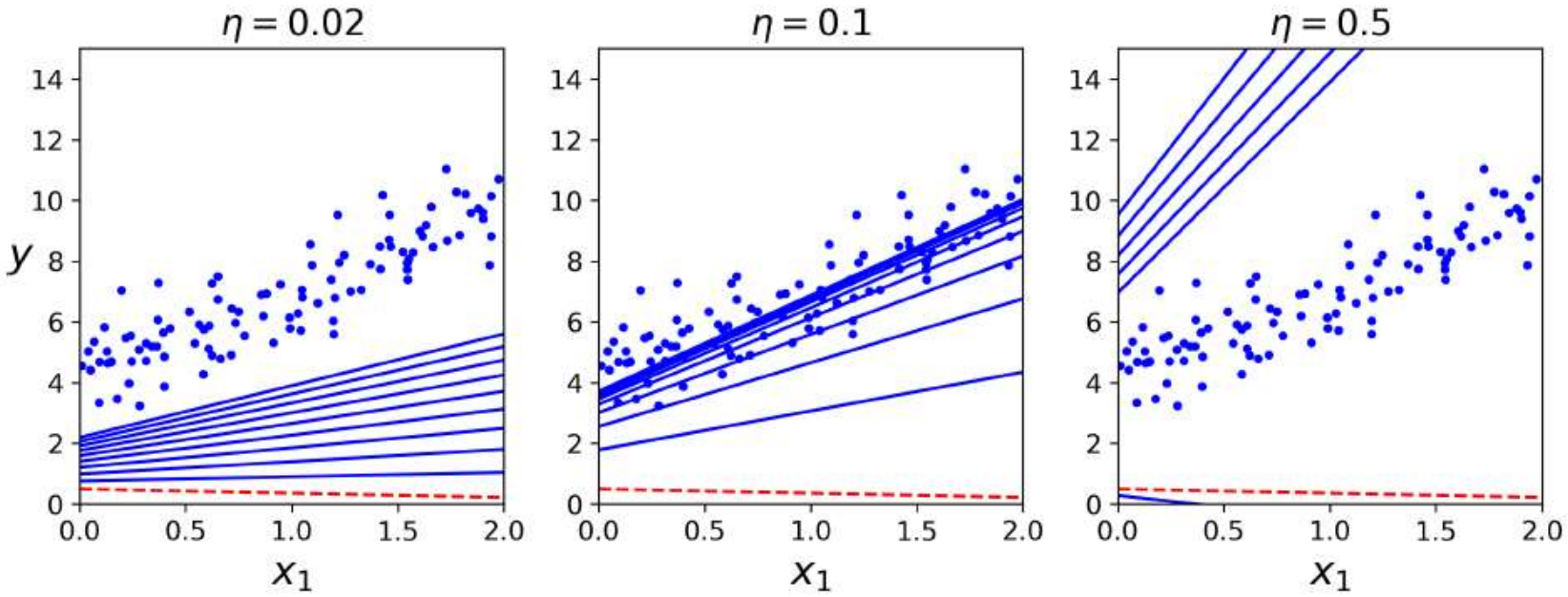
124 Gradient Descent Step 크기 결정

Equation 4-7. Gradient Descent step

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

- Eta : Learning rate
- Colab

125 Learning rate



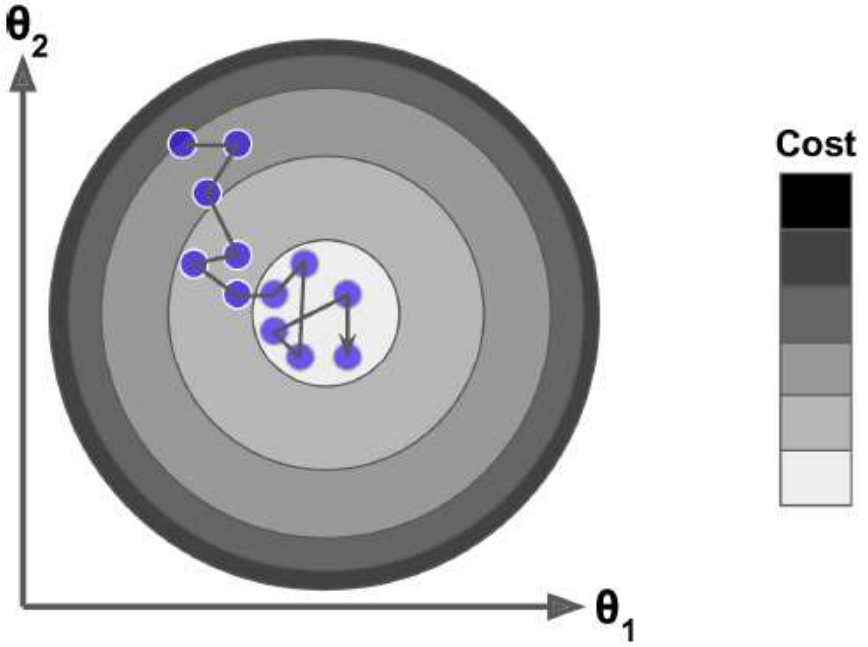
좌측 : 너무 낮은 Learning rate : 시간이 많이 걸림
중간 : 적절함. 몇번의 시도 끝에 이미 솔루션에 도달
우측 : 너무 높은 Learning rate

125~126 적절한 Learning rate를 어떻게 찾는가?

- grid search로 찾으면 됨
  - 반복횟수에 따라 시간이 많이 걸릴 수 있음.
- 반복횟수는 어떻게 정하는가?
  - 너무 적으면 최적값에서 멀 수도 있음.
  - 너무 많으면 시간낭비 할 수 있음.
  - -> 방법은? 반복횟수를 많이 설정하되, gradient vector가 매우 적어지면(공차 이하로 떨어지면) 해당 알고리즘을 interrupt 하는것.
  - 공차를 1/10으로 줄이면, 구하는데 걸리는 시간은 10배 더 걸릴 수 있음.

126 Stochastic Gradient Descent

- Batch Gradient Descent의 가장 큰 문제는 매 스텝마다 전체의 배치를 사용하는것임. -> 트레이닝 셋이 클 경우 매우 느려짐.
- Stochastic Gradient Descent는 매 스텝에 랜덤한 instance를 골라 경사를 계산함. 알고리즘을 매우 빠르게 할 수 있음. 매우 큰 트레이닝 셋에서도 잘 됨.
  - 특징 : Batch Gradient Descent에 비해 less regular(비정규적)임. = cost function이 작아졌다 커졌다 반복됨. 평균적으로는 감소. 그러나 미니멈 포인트에서 가서도 왔다갔다함. 절대 안정화되지 않음. 즉, 알고리즘이 멈췄을때의 마지막 파라미터는 good 하지만, 최적(optimal)은 아니다.



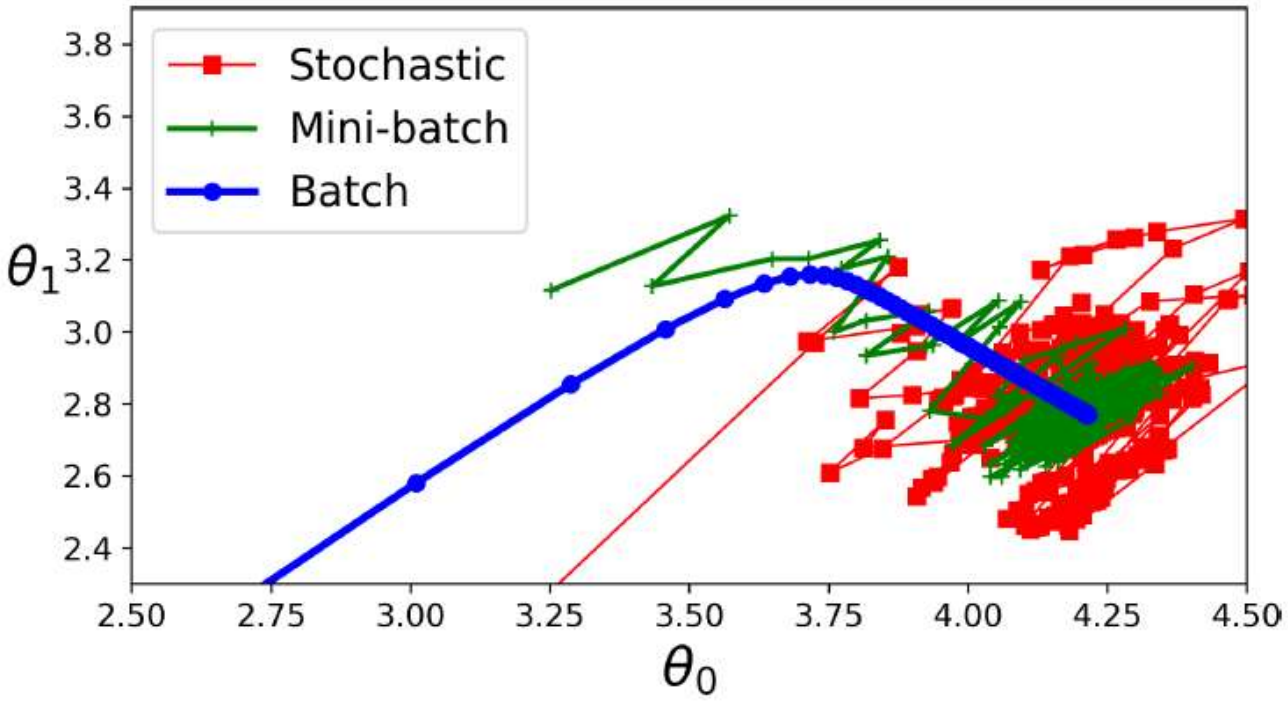
127 Stochastic Gradient Descent

- Cost function이 very irregular(비정규적)한 경우, SGD는 local minima를 벗어나는데 도움이 됨 ->Batch Gradient Descent 보다 global minimum을 더 잘 찾을 수 있음.
- 랜덤성은 local minima로부터 벗어나는것에는 도움이 되지만, 정말 최소값에서는 안정화 될 수 없음.
- -> 해결법 : learning rate를 점진적으로 낮추는것.
  - learning schedule 이라고 불리는 function을 사용하면 매 회에서의 learning rate를 알 수 있음.
  - Learning rate가 너무 빨리 감소하면, local minimum을 최소값이라 착각하게 될 수도 있음. 혹은 최소값에 가기도 전에 종료될 수도 있음.
  - 너무 느리게 감소하면 시간이 많이 걸릴 수 있음. 이 경우 트레인을 너무 일찍 종료해도 최소값이 아닌곳에서 끝날 수 있음.
- Colab

129 Mini-batch Gradient Descent

- Mini-batch 라 불리는 small random sets of instances(작은 랜덤셋)의 경사를 계산함
- 장점 : GPU사용시 SGD보다 나은 퍼포먼스를 낼 수 있음. SGD보다 더 나은 최소값을 가질 수 있음.
- 단점 : SGD보다 local minima 를 벗어나기가 어려움.

129 각 GD별 최소값 도달형태



언뜻 보면 Batch가 가장 나아 보일 수 있지만, 시간이 굉장히 많이 걸림.
Mini-batch GD와 SGD도 learning schedule만 적절하게 조절되면 최소값을 구할 수 있음.

130 GD 알고리즘 비교

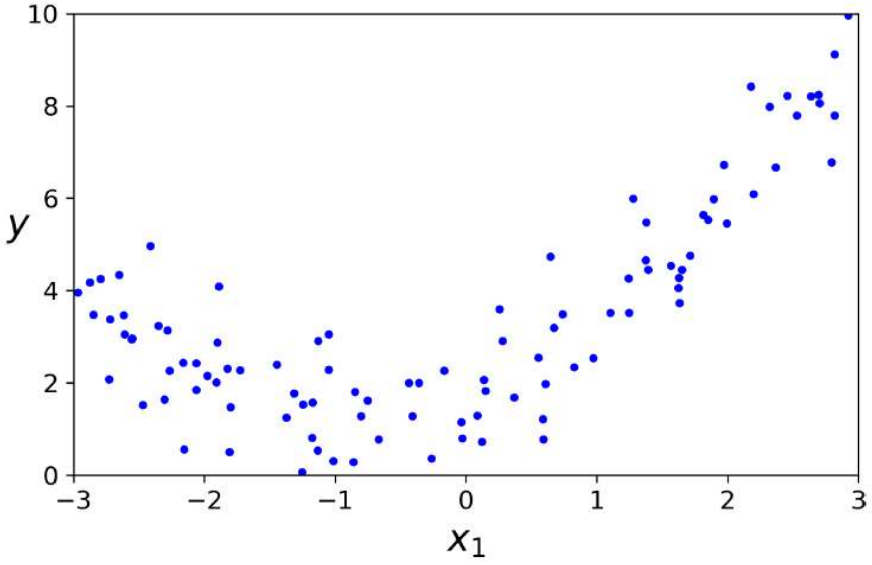
Algorithm	Large *m*	Out-of-core support	Large *n*	Hyperparams	Scaling required	Scikit-Learn
Normal Equation	Fast	No	Slow	0	No	n/a
SVD	Fast	No	Slow	0	No	`LinearRegression`
Batch GD	Slow	No	Fast	2	Yes	`SGDRegressor`
Stochastic GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`
Mini-batch GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`

m : instance의 수
n : feature의 수

Training이 끝난 후에는 거의 차이점이 없으며, Predict도 정확히 같은 방법으로 도출해냄

130~131 Polynomial Regression

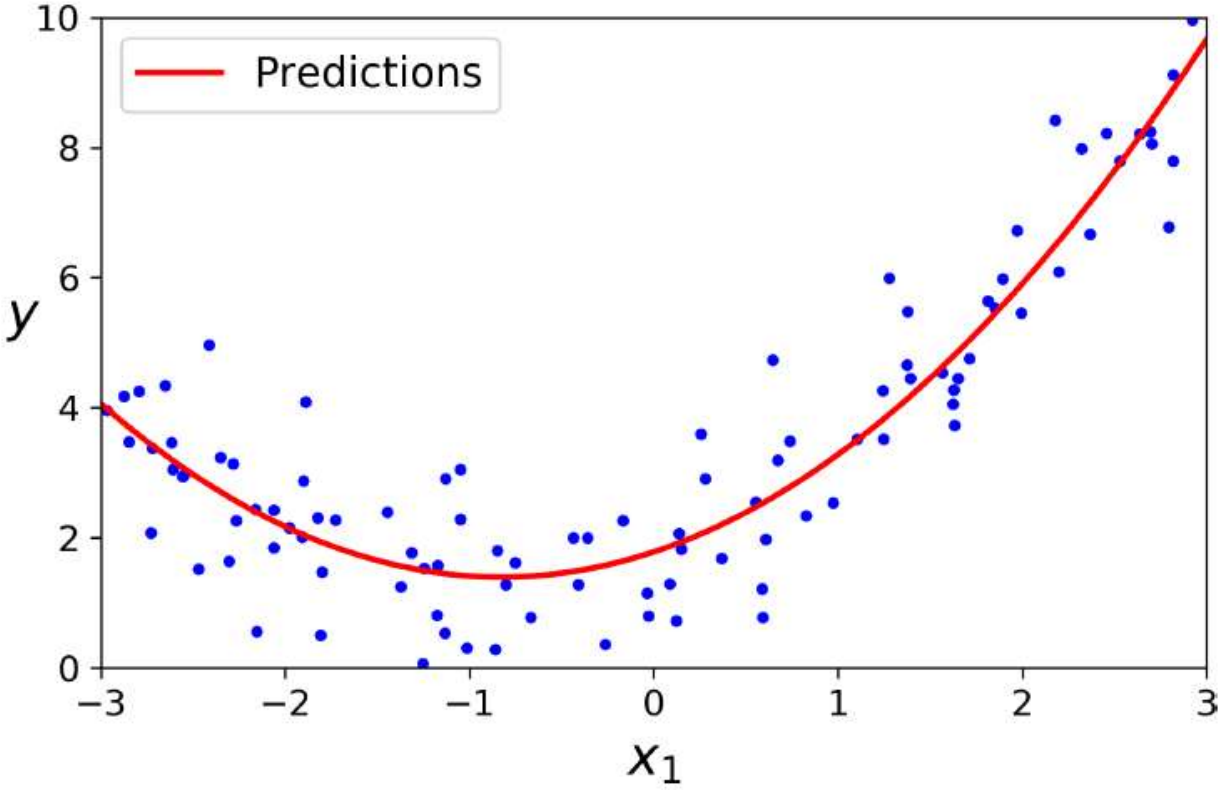
- 비선형 데이터를 선형 모델로 트레인 시킬 수 있음.
- 간단한 방법은 각 feature에 거듭제곱을 추가하여 새로운 feature를 만드는것임. 그리고 그 새로운 feature을 트레인 하는것.
  - 이 방법을 Polynomial Regression이라 부름.



직선으로는 좌측과 같은 데이터를 제대로 트레인 할 수 없음.
2차원에서는 제곱을 하여 새로운 feature를 만듦.

colab

132 Polynomial Regression model predictions



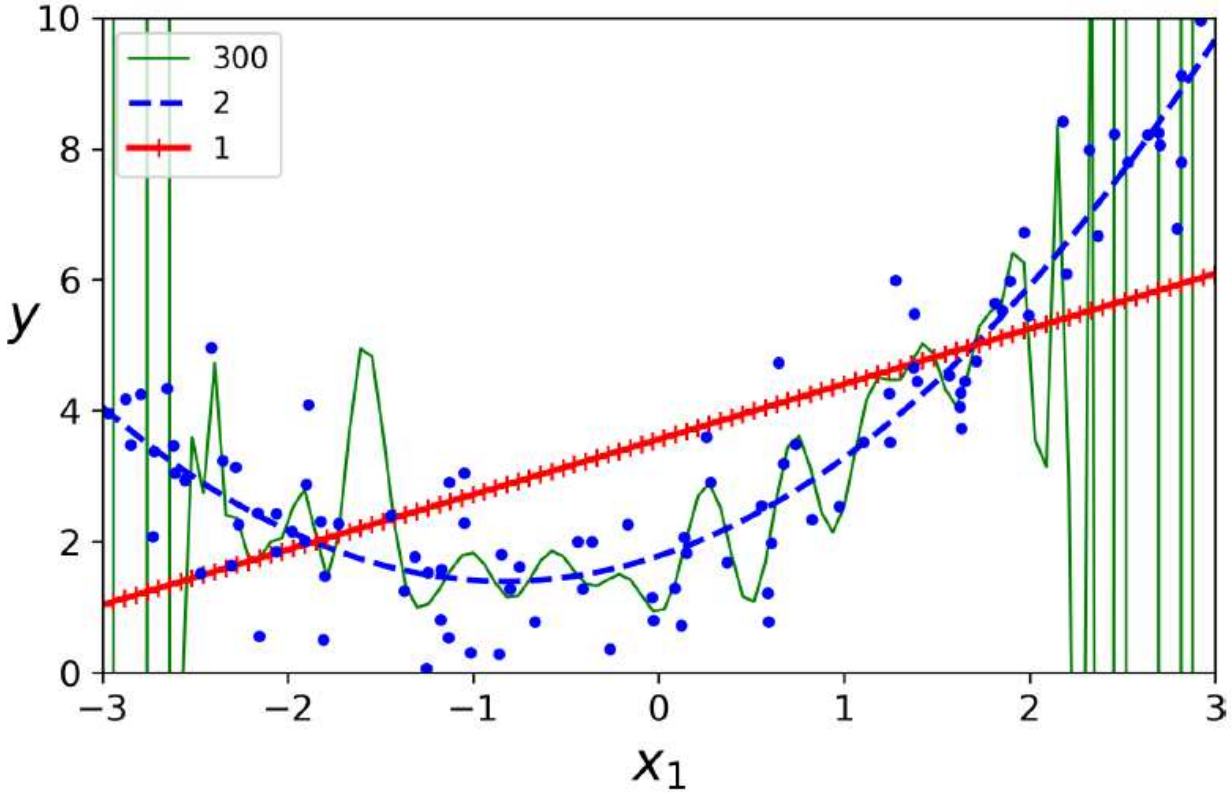
여러가지 feature들이 있을 때,
Polynomial Regression은 각 feature간 관계를 찾을 수 있음.
PolynomialFeatures라고 불리는 기능에 의해 가능함.
Ex. a와 b라는 feature가 있을때,
PolynomialFeatures=3을 한다면
$a^2, a^3, b^2,$ and b^3
라는 feature만 추가하는게 아니라,
$ab, a^2b,$ and ab^2.
도 함께 주가함.

132 PolynomialFeatures(degree=d) + Learning Curves

- PolynomialFeatures(degree=d)는 n개의 feature를 가지고 있는 어레이를 $\frac{(n+d)!}{d!\,n!}$ 개의 feature를 가지고 있는 어레이로 변환함.

- Learning Curves
  - 고차원의 Polynomial Regression 을 진행 할 경우, plain Linear Regression보다 훨씬 더 train이 잘 된다.

133 High-degree Polynomial Regression



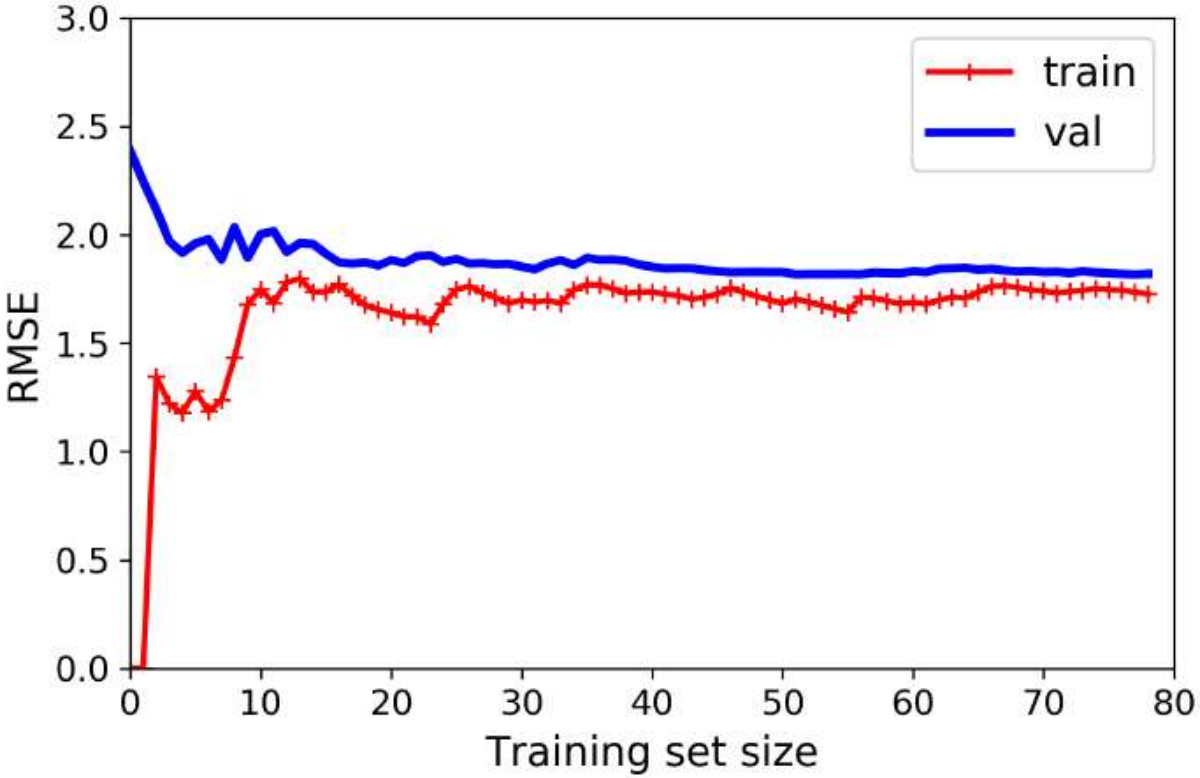
좌측 polynomial Regression model은 굉장히 overfit됨.
반면 linear model은 underfit됨.
좌측 상황에서 가장 잘 맞는 데이터는 quadratic model임.

-> 그럼 어떤 모델을 사용하여야 할지 어떻게 알 수 있나?
1. Train 데이터에는 잘 맞지만, cross-validation metrics에서 잘 맞지 않을 경우 overfit되어있음.
2. train 데이터와 cross-validation metrics 에서 모두 맞지 않으면, underfit되어있음.
3. Learning curves를 확인하는방법.

133 learning curves

- training set and the validation에서의 해당 model의 performance plot임.
- 방법 -> 그냥 다른 사이즈의 트레이닝 세트를 여러번 트레인 하는것.
- Colab

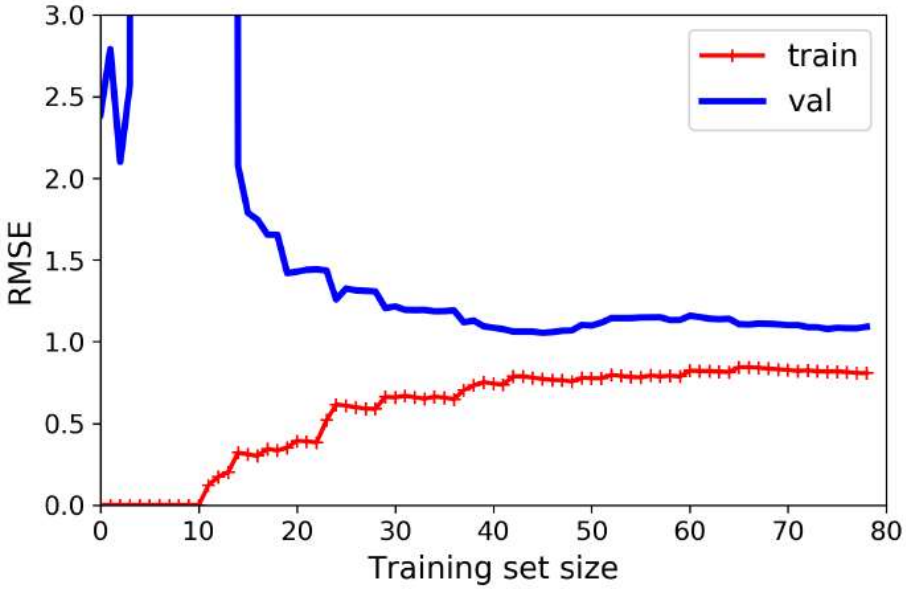
134 learning curves



트레이닝 세트가 1~2개밖에 없을 경우 굉장히 train이 잘 된 모습임.-> 커브가 0에서 시작함. 다만 generaliz하기에는 부족한 모습을 보임(validation error가 높음)
새로운 instance들이 늘어남에 따라, 점점 데이터는 더 안맞게 되고 결국 plateau에 도달하게 됨. -> instance를 늘려도 큰 차이가 없어짐.
해당 모델은 underfitting상태임. -> Train과 validation error가 둘다 높으면서 서로 근접함.
만약 모델이 underfitting 상태 일 경우, training example을 늘린다고 해도 큰 차이가 없을것임. -> 좀 더 복잡한 모델을 사용하거나 better featur를 사용하여 해결

135~136 10차원 polynomial model의 learning curves(위와 같은 데이터)

- Colab



위(Linear Regression model)와 가장 큰 차이점 2가지

1. Training data error가 훨씬 낮음.
2. 두 커브 사이에 gap이 있음. -> training data error가 훨씬 적다 : 훨씬 잘 맞다. = overfitting. 그러나 데이터 사이즈를 늘릴수록 두 커브가 좁혀짐.
3. Overfitting모델을 개선시키는 방법은 train data를 더 크게 하는것. (validation error가 training error에 도달 할 때 까지)

136 The Bias/Variance Tradeoff

- 모델의 일반화 에러는 3가지 다른 에러의 합으로 나타낼 수 있다.
- 1. Bias
  - 틀린 추측에 의해 발생함. (ex. 데이터가 2차원인데 1차원이라고 생각한경우) – underfit 되는경우가 많음.
- 2. Variance
  - 모델의 민감도에 의해 발생. 작은 변화에도 민감하게 발생하는 모델일 경우. (ex. 고차원의 polynomial model 같은 경우) – overfit 되는경우가 많음.
- 3. Irreducible error
  - 데이터 자체의 노이즈 때문에 발생. 유일한 해결책은 해당 데이터를 지우는것. (ex. Outliar)
- 모델이 복잡해 질 수록 variance은 증가하고, Bias는 줄어듬.
- 반대로, 모델이 간단해 질 수록 Bias는 증가하고 Variance는 줄어듦.
- 위때문에 Tradeoff라고 불림.

136 Regularized Linear Models

- Overfitting을 줄이는 방법으로는 regularize하는것임.
- DOF(degree of freedom)이 줄어들수록 overfit되지 않음.
- 간단한 예로 Polynomial model을 regularize 하려면 polynomial degree를 줄이면 됨.
- Linear model의 경우, 모델의 weight를 규제하면 됨.
- Ridge Regression, Lasso Regression, and Elastic Net을 사용하면 weight를 규제 할 수 있음.

137 Ridge Regression(=*Tikhonov regularization)*

- Linear Regression의 regularized된 버전임.

  a *regularization term* equal to $\alpha \sum_{i=1}^{n} \theta_i^2$

- 을 추가하는건데, 이건 learning algorithm 이 데이터를 train할 수도 있게 만들지만, model weight을 가능한 한 작게 만들기도 함.
- Training 중 사용되는 Cost function과 testing을 위해 사용되는 performance measure는 다르다. Cost function 은 최적화와 친근한 미분값이고, performance measure은 최종값과 가장 가까운 값임.
  - Ex. Classifier (cost function : log loss / evaluation : precision, recall)

137 Ridge Regression(=*Tikhonov regularization)*

- α *:* 얼마나 regularize 하기를 원하는가. $\alpha = 0$ 일 경우, ridge regression은 그냥 linear regression과 같음.
- α이 매우 클 경우, 모든 weight가 0에 가깝게 되며, 결과는 데이터의 평균으로 가는 flat line(평평한 선)이 됨.

Equation 4-8. Ridge Regression cost function

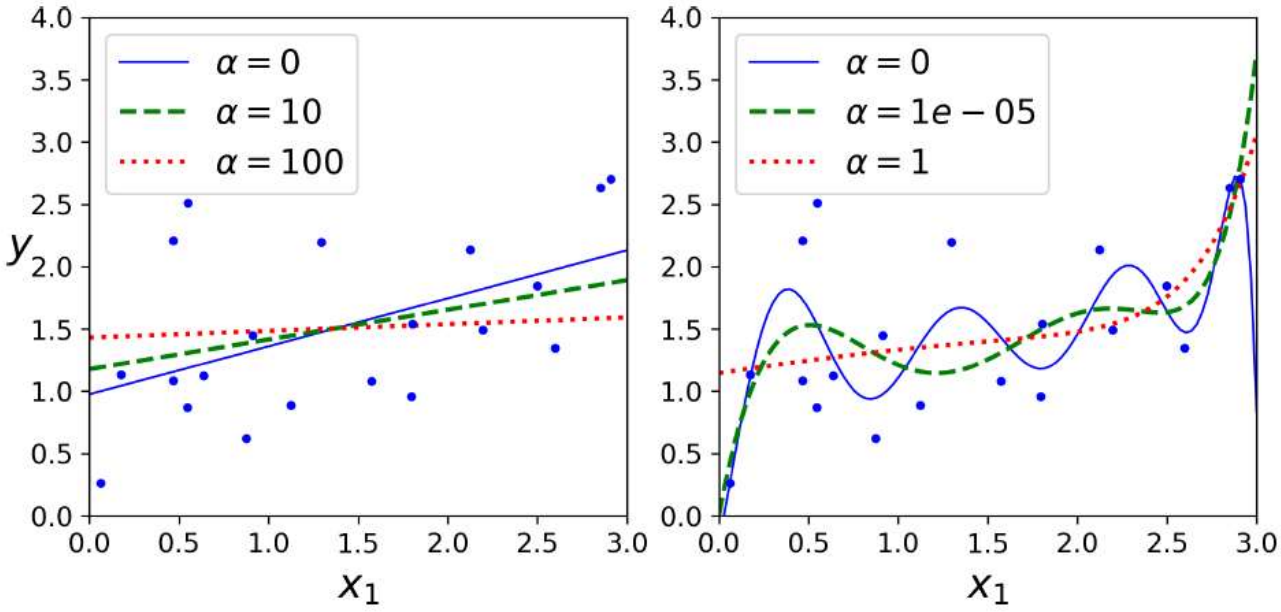
$$J(\mathbf{\theta}) = \mathrm{MSE}(\mathbf{\theta}) + \alpha\frac{1}{2}\sum_{i=1}^{n}\theta_i^2$$

- θ_0 는 regularized되지 않음. 만약 w를 feature weight vector라 하면, 윗 식은 $\frac{1}{2}(\|\mathbf{w}\|_2)^2$ 로 나타내질 수 있음.

137 Ridge Regression(=*Tikhonov regularization)*

- Ridge Regression을 하기 전에, 데이터를 scale하는게 중요함.
  - ex. StandardScaler
  - Input feature에 민감함.
  - 거의 모든 regularized model에 적용됨.

138 Ridge Regression(=*Tikhonov regularization)*



좌측 : Plain Ridge Model 이 사용됨.
우측 : PolynomialFeatures(degree=10)로 먼저 데이터를 늘리고, StandardScaler로 scale작업을 한 후, Ridge model이 적용됨.

Closed-form solution을 이용하여서도 ridge regression을 할 수도 있음. 혹은 Gradient Descent 를 이용해도 할 수 있음.

Equation 4-9. Ridge Regression closed-form solution

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

Colab

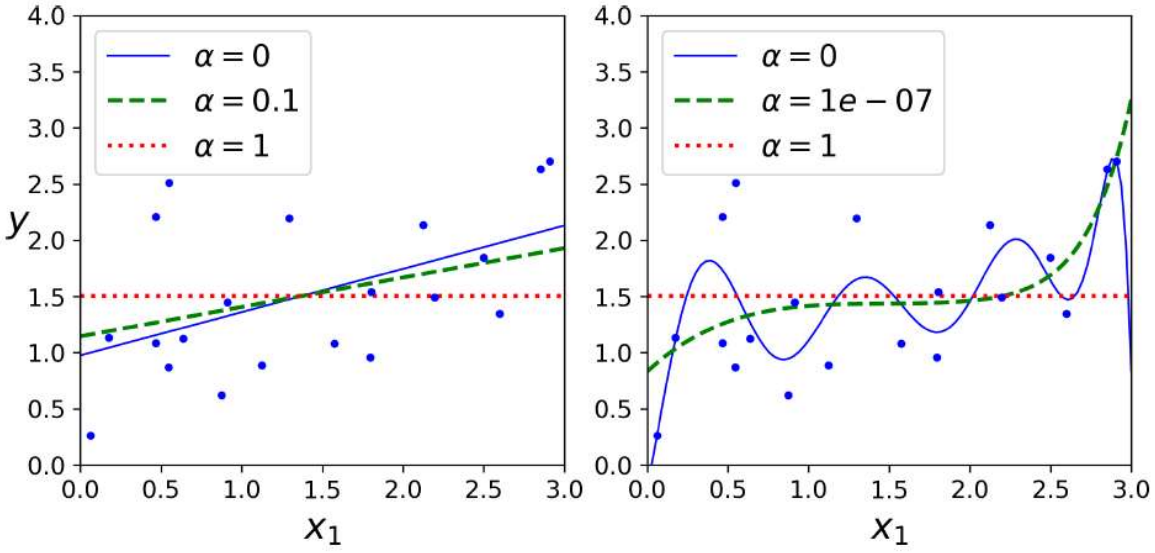
139 Lasso Regression(*Least Absolute Shrinkage and Selection Operator Regression)*

- Linear Regression의 또다른 regularized 버전임.
- Cost function에 regularization term을 추가함.
- 또 다른 특징은 half the square of the ℓ2 norm 대신에 ℓ1 norm of the weight vector를 사용함.

Equation 4-10. Lasso Regression cost function

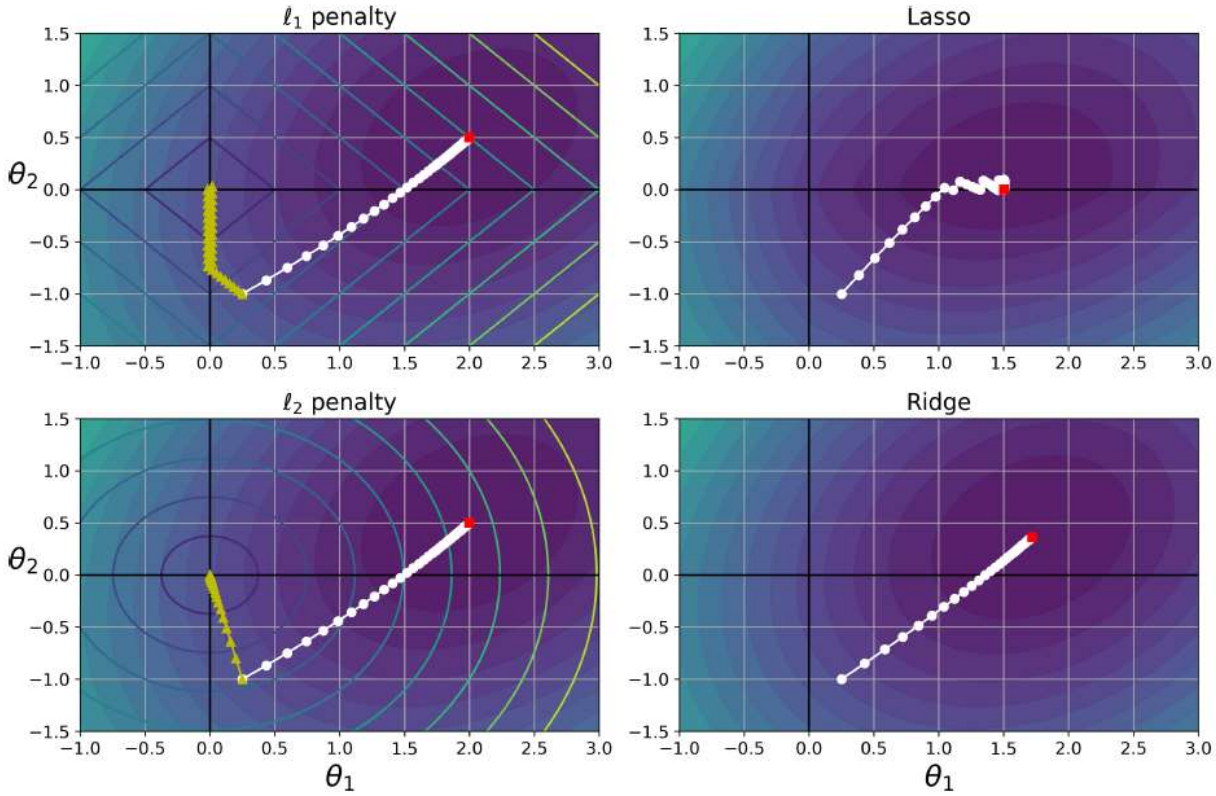
$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \alpha \sum_{i=1}^{n} |\theta_i|$$

140 Lasso Regression(*Least Absolute Shrinkage and Selection Operator Regression)*



중요한 특징 : 중요성이 가장 낮은 Feature들의 weight를 없앰.(ex. 0으로 바꿈) 우측그래프의 1e-07 의 선을 보면, 거의 선형임.고차원의 feature들을 0으로 바꿔준것. 즉, 자동으로 중요한 feature들을 고르고 sparse model을 내보냄(0이 아닌 feature들)

140~141 Lasso / Ridge Regression에서 ℓ1 / ℓ2 패널티가 미치는 영향

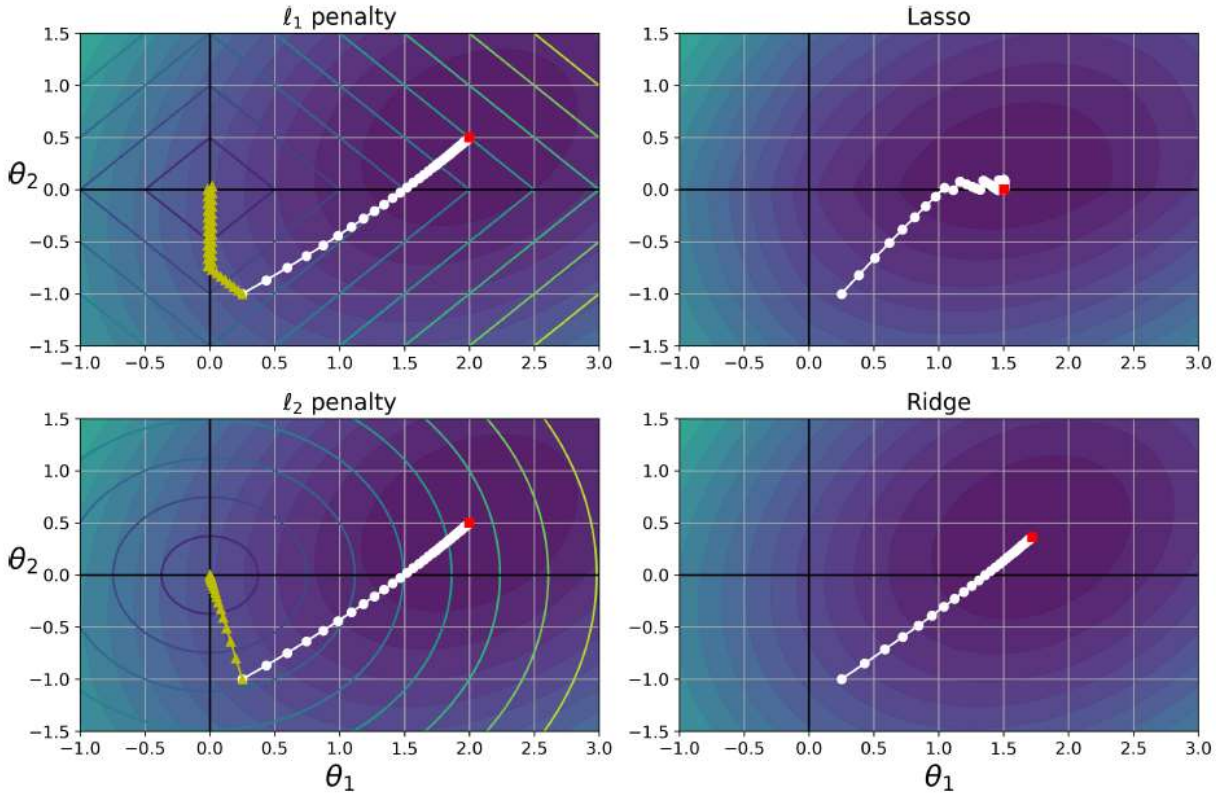


상단 그림만 먼저 설명
좌상단 그림의 타원 : unregularized MSE cost function(α = 0)
흰색 원 : Batch Gradient Descent
좌상단 그림의 다이아몬드 : ℓ1 penalty
삼각형 : BGD path for this penalty only ($\alpha \rightarrow \infty$). 패널티.
ℓ1 penalty의 경우, θ1 = 0가 먼저 된 후에 ϑ2 = 0가 될 때 까지 줄어듦.
우측 상단 : 같은 cost function이지만, ℓ1 penalty with α = 0.5 를 추가한것임. Global minima는 ϑ2 = 0 axis에 있음.
BGD 가 먼저 ϑ2 = 0에 도달하면 global minimum으로 감.

하단 그림
위 조건과 동일하나 ℓ2패널티를 사용함.

Regularized되면 더 global minimum에 근접함. 그러나 weight들은 완전히 사라지지 않을 수 있음.

141 Lasso / Ridge Regression에서 ℓ1 / ℓ2 패널티가 미치는 영향



우측 상단 그림에서 BGD가 왔다갔다 하는 이유는 $\vartheta 2 = 0$에서 slope가 굉장히 많이 바뀌기 때문임.

Lasso cost function은 $\vartheta 2 = 0$에서 미분 불가능하지만, *subgradient vector* **g**15를 사용한다면 진행가능함.

Equation 4-11. Lasso Regression subgradient vector

$$g(\boldsymbol{\theta}, J) = \nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta}) + \alpha \begin{pmatrix} \operatorname{sign}(\theta_1) \\ \operatorname{sign}(\theta_2) \\ \vdots \\ \operatorname{sign}(\theta_n) \end{pmatrix} \quad \text{where} \quad \operatorname{sign}(\theta_i) = \begin{cases} -1 & \text{if } \theta_i < 0 \\ 0 & \text{if } \theta_i = 0 \\ +1 & \text{if } \theta_i > 0 \end{cases}$$

Colab

142 Elastic Net

- Ridge Regression 과 Lasso Regression 사이임.
- Regularization term은 Ridge와 Lasso를 섞은것임.
  - Mix ratio r 도 변경가능.
  - r=0 이면 Ridge, r=1이면 Lasso로 동작.

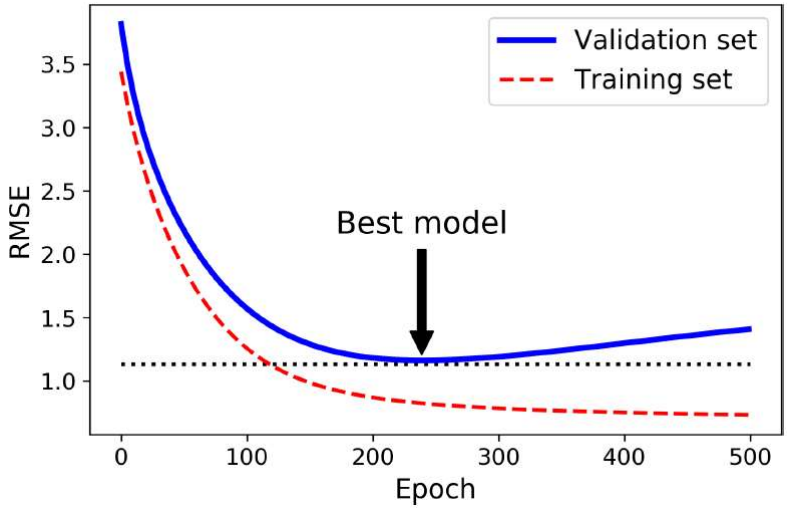
Equation 4-12. Elastic Net cost function

$$J(\mathbf{\theta}) = \text{MSE}(\mathbf{\theta}) + r\alpha\Sigma_{i=1}^{n}\left|\theta_i\right| + \frac{1-r}{2}\alpha\Sigma_{i=1}^{n}\theta_i^2$$

- 그렇다면 언제 plain Linear Regression, Ridge, Lasso, Elastic Net등을 써야할까?
  - 약간의 regularization은 거의 항상 좋음. 그러니까 regularization을 하지 않는 plain Linear Regression은 피하는게 좋다.
  - Ridge는 괜찮긴 하지만, 몇가지의 Feature만 중요하다 생각된다면 Lasso나 Elastic Net이 좋음.
  - Feature의 수가 instance의 수보다 클 경우,혹은 몇가지의 feature들이 서로 강하게 연결돼 있는 경우에는 Elastic Net이 Lasso나 Ridge보다는 선호됨.
- Colab

142~143 Early Stopping

- GD시리즈와 같은 알고리즘을 정규화 하는 방법은, validation error가 최소에 도달했을 때, training을 중지시키는것임.
  - *early stopping* 이라 부른다.



고차원의 Polynomial Regression Model임. BGD로 트레인되었음. 트레이닝을 진행하면 Validation / Training에 관한 RMSE는 줄어듦. 근데 그 후에도 계속하다보면 Validation Error는 늘어나는 현상이 발생함.(Overfit)
Early stopping을 이용하여 Best model일때 트레인을 멈추게 할 수 있음.

SGD나 Mini-batch GD는 선이 완만하지 않을수도 있음. 최소값에 도달했는지 아닌지도 애매한 경우가 있음. 해결책은 validation error가 어느정도의 시간동안 minimum 일 경우, 최소의 minimum인 상태로 되돌려서 그때의 파라미터로 세팅을 하는 것.

Colab

144 Logistic Regression

- 몇가지의 regression 알고리즘은 classifier로도 사용 될 수 있음.
- Logistic Regression은 instance가 어떤 특정 클래스에 속할 가능성을 알아내려 할때도 쓰임.
  - 만약 가능성이 50%가 넘는다면 그 해당 instance가 그 class에 속한다고 여김. -> binary classifier이 됨.
- Linear Regression과 같이, Logistic Regression또한 input feature들의 weight 합계를 계산함. (bias도 함께) 근데 차이점은, Linear regression과는 다르게 이것은 *logistic*을 출력함.

144~145 Logistic function

Equation 4-13. Logistic Regression model estimated probability (vectorized form)

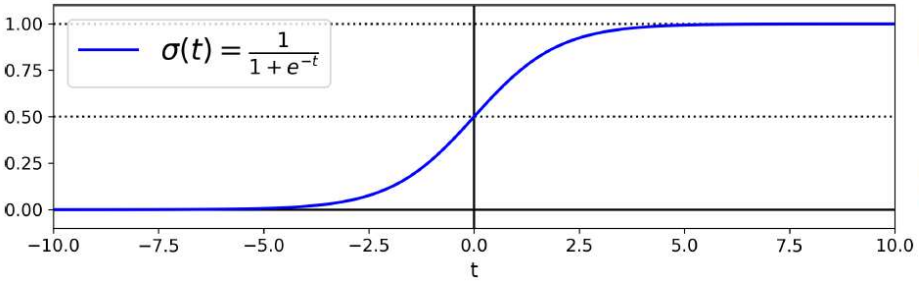
$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma\left(\mathbf{x}^T\boldsymbol{\theta}\right)$$

$\sigma(\cdot)$—is a *sigmoid function*

Sigmoid function은 출력으로 0 과 1 사이의 값을 내보냄.

Equation 4-14. Logistic function

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



Equation 4-15. Logistic Regression model prediction

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

145~146 Logistic Regression - Training and Cost Function

- Training 의 목적 : parameter vector **θ**를 세팅하는것.
  - positive instances (y =1) 이 높은 가능성을 가지게 만들고,
  - negative instances (y = 0) 이 낮은 가능성을 가지게 만드는것.

Equation 4-16. Cost function of a single training instance

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

- t가 0으로 갈수록 $-\log(t)$은 매우 커진다.
  - -> positive instance에서 가능성이 0에 가까워질수록 cost 가 매우 커짐.
  - -> negative instance에서 가능성이 1에 가까워질수록 cost가 매우 커짐.
  - 가능성이 negative 에서 0, positive 에서 1일수록 cost가 0에 가까워짐.

146 *log loss*

Equation 4-17. Logistic Regression cost function (log loss)

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

전체 트레이닝 셋의 cost function은 전체 training instance의 평균임.
위와 같은 식으로도 나타 낼 수 있음. 이것은 log loss라 부름.
최소값의 θ를 구할 수 있는 closed-form equation 이 없음. 그러나 위 cost function은 convex(볼록한형태)이기 때문에, GD시리즈로 확실하게 global minimum을 찾을 수 있음.

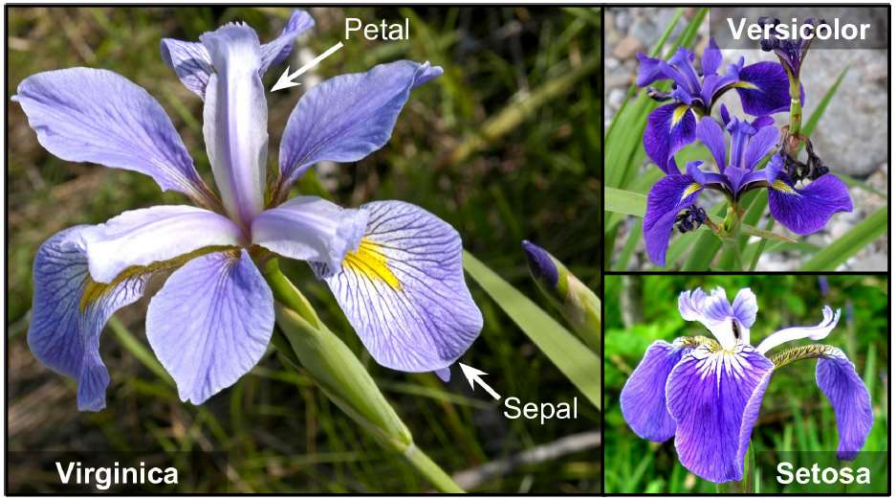
j번째 모델 파라미터 ϑj 의 cost function의 partial derivatives(편미분) 은 아래와 같음.

Equation 4-18. Logistic cost function partial derivatives

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

각 instance들은 prediction error을 구하고, j번째 feature value를 거기에 곱한다. 그리고 전체 평균을 구한다. 전체 gradient vector를 구하기만 한다면, BGD 알고리즘을 사용 가능함.
이것이 Logistic Regression model을 트레인 하는 방법.
SGD같은 경우는 한번에 하나의 인스턴스에서, mini-batch GD같은 경우는 한번에 하나의 mini-batch에서 사용가능.

146~147 Decision Boundaries



Logistic Regression을 iris dataset에 적용해보자.

colab

148 Decision Boundaries

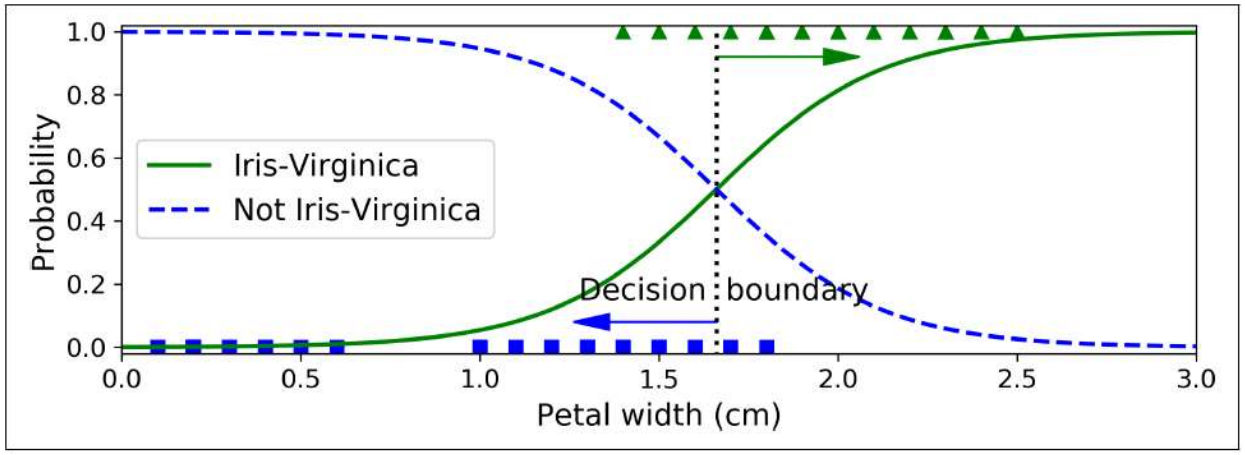


Figure 4-23. Estimated probabilities and decision boundary

Iris-Virginica의 Petal width는 1.4cm에서 2.5cm까지임.
다른종의 petal width는 0.1cm에서 1.8cm임.
2cm가 넘는 구간에서는 classifier가 해당 꽃은 Iris-Virginica라고 거의 확신함.
1cm보다 아래 구간에서는 그 반대임.
그 사이 구간에서는 애매하다.
1.6cm구간에 있는 경계를 Decision boundary라고 함.(양쪽 다 확률이 50%정도)
= 1.6cm보다 petal width가 높으면, classifier는 해당 꽃은 Iris-Virginica라고 예측할것임.

148~149 Decision Boundaries

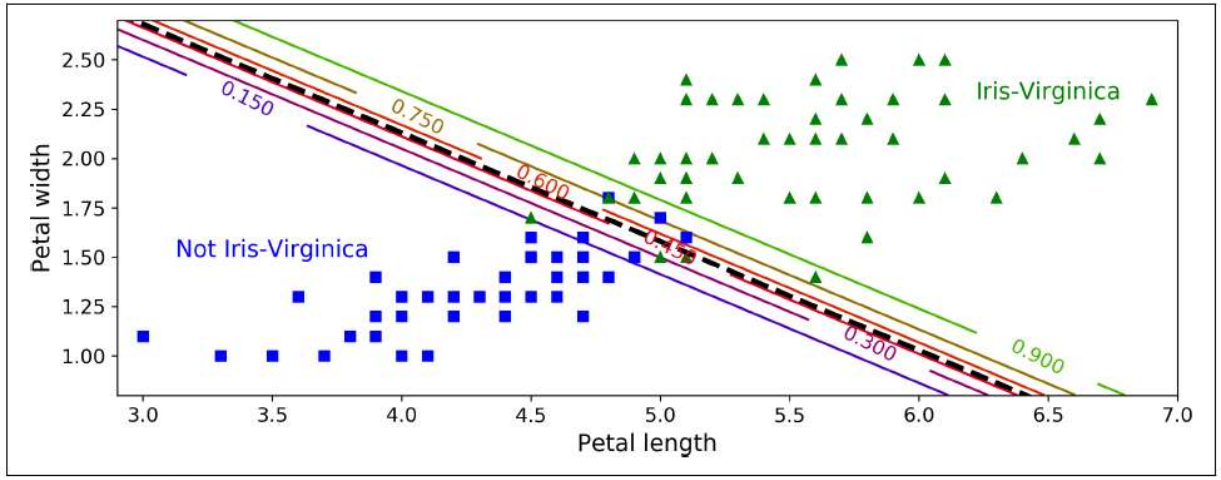


Figure 4-24. Linear decision boundary

다른 linear model들과 같이, Logistic Regression model도 ℓ1 or ℓ2 penalties 를 이용하여 regularized 될 수 있음. (Scikit-Learn은 디폴트값으로 ℓ2 패널티가 설정 돼 있음)

전 페이지와 같은 데이터셋임.
현재는 2개의 feature를 나타냄. Width 와 length.
Logistic Regression classifier가 train이 되면, 2개의 feature에 대해서도 estimate 할 수 있음.
중간지점의 dashed line은 50%의 가능성임.
-> 현재 모델의 Decision Boundary임.
Linear형태임.
그외에도 parallel line은 특정 확률을 나타냄(15%~90%) -> 해당 line 윗쪽의 모든 꽃들은 Iris-Virginica일 확률이 15~90%임.

149 Softmax Regression(= *normalized exponential)*

- Logistic Regression은 트레인 한 후에 다른 multiple binary classifiers 와 결합하지 않고도 여러개의 class로 바로 일반화 될 수 있다.
  - 즉 Logistic Regression이 binary 말고도 multiple 에도 적용 가능하단뜻
- 이것은 *Softmax Regression*, 또는 *Multinomial Logistic Regression* 이라 불림
- instance **x**가 주어졌을 때, Softmax Regression은 먼저 각 클래스 k별 스코어 $s_k(\mathbf{x})$를 먼저 계산한다. 그 후에 각 클래스별 확률을 score에 softmax function을 적용하여 계산함.
- 스코어 구하는 방법은 Linear Regression prediction와 형태가 비슷함

Equation 4-19. Softmax score for class k

$$s_k(\mathbf{x}) = \mathbf{x}^T \boldsymbol{\theta}^{(k)}$$

149~150 Softmax Regression(= *normalized exponential)*

- 각 클래스는 parameter vector **θ***(k)*를 가지고 있음.

Equation 4-20. Softmax function

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

K : 클래스 수

s(x) : instance x에 대해 모든 클래스의 스코어를 가지고 있는 벡터

$\sigma(\mathbf{s}(\mathbf{x}))_k$: 모든 스코어를 고려했을때 instance x가 k클래스에 속할 확률

- 스코어 $s_k(\mathbf{x})$ 를 계산 한 후에 가능성 계산 가능함.
  - 해당 instance가 *k*라는 클래스에 속할 확률
  - 모든 스코어의 exponential을 계산하여 normalize 함(dividing by the sum of all the exponentials = 모든 exponential의 합으로 나눔)
- 스코어는 보통 logits 또는 log-odds 라 불림.

150 Softmax Regression(= *normalized exponential)*

- Logistic Regression classifier처럼 Softmax Regression classifier도 역시 가장 높은 가능성을 가진것을 클래스로 예측함.

Equation 4-21. Softmax Regression classifier prediction

$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\operatorname{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}}\ \left(\left(\boldsymbol{\theta}^{(k)}\right)^T \mathbf{x}\right)$$

- *argmax* 연산자는 해당 펑션을 최대화하는 변수값을 return한다.
- 위의 argmax는 k값을 리턴하는데, 이 k값이 $\sigma(\mathbf{s}(\mathbf{x}))_k$.(가능성)을 최대화하는값임.

150~151 Softmax Regression(= *normalized exponential) / cross entropy*

- Softmax Regression classifier는 한 번에 하나의 값만 예측함.
  - i.e., it is multiclass, not multioutput
  - 따라서 이 분류기는 상호배타적인 클래스(예. 다른 종류의 식물)에만 사용됨. 한 사진에서 여러명의 얼굴을 찾는것에는 사용될 수 없음.

Equation 4-22. Cross entropy cost function

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

- 위의 공식은 예측된 가능성이 얼마나 잘 들어맞냐를 측정할때 사용함.
- $y_k^{(i)}$: i번째 인스턴스가 k클래스에 속할 가능성. 보통 0 또는 1임
  - 만약 k가 2라면(즉 클래스가 2개밖에 없다면) Logistic Regression's cost function과 같음.

151 Cross Entropy

Equation 4-23. Cross entropy gradient vector for class k

$$\nabla_{\theta^{(k)}} J(\mathbf{\Theta}) = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{p}_k^{(i)} - y_k^{(i)} \right) \mathbf{x}^{(i)}$$

o o