6️⃣

ML06 (5-8)

The Bias/Variance Tradeoff

- 모델 일반화 에러: 3개의 다른 에러의 합으로 표현된다

1. **Bias (편향)** : 모델의 예측이 실제 데이터와 얼마나 일치하는지에 대한 편견 or 오차
    - ex. 2차원(quadratic)을 1차원(linear)로 잘못 추정)
    - coefficient = intercept와 다름. (y = ax + b에서 b를 bias라고 하니까 헷갈렸음)
    - **높은 bias = 거의 UNDERFIT**
2. **Variance (분산)** : 학습 데이터의 작은 변화에도 과도하게 민감함
    - 자유도(degrees of freedom)가 높으면 **high variance → OVERFIT**
        - 자유도 = 매개변수 수 (a_0, a_1, … a_n)
        - 자유도가 높다. = 높은 차수의 다항식 모델(polynomial model)
            - y = a_0 + a_1*x + a_2*x^2 + … + a_n*x^n
    - 민감하다 = 새로운 데이터에 대한 일반화 능력이 낮다. (학습 데이터에 변화가 생기면 예측 결과가 크게 흔들린다.)

3. **Irreducible Error** (불가항력 오차; 모델로는 완전히 제거 불가능한 오차) : 데이터 자체에 노이즈 있음
    - 해결방안: 데이터를 정제한다 (데이터를 고치거나, 특이값(outliers)을 감지하고 제거한다.)

> 💡 **Tradeoff** → model complexity를 늘리면, variance가 올라가고, bias는 낮아진다. (overfit)

5. Regularized Linear Models

지금 linear 모델에 대해 공부하고 있으니까, linear model에 대한 regularization을 살펴보자.

- **Regularization** : 과적합(Overfitting)을 줄이는 데 사용되는 효과적인 방법 중 하나
    - 선형 모델(Linear Model)의 가중치(Weights)를 제한하여 모델의 복잡성을 줄임
    - linear model에 적용할 수 있는 3가지 regularization
        1. Ridge Regression
        2. Lasso Regression
        3. Elastic Net

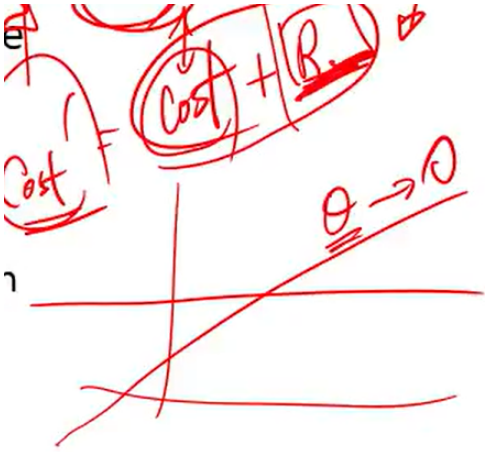
1. Ridge Regression (= L2 regularization = Tikhonov regularization)

- linear에 ridge regularization을 적용한 것
- 훈련(training) 시, 원래의 손실 함수(cost function; (R)MSE)에 가중치(Weights)의 제곱 합에 비례하는 정규화 항을 추가

$$\alpha \sum_{i=1}^{n} \theta_i^2$$

- 모델의 가중치 값을 작게 유지하도록 강제함.
- 학습하는 과정에서만 적용됨. 성능 측정 시에는 Unregularized 상태로 평가함.

- *α :* 정규화 정도를 조절하는 하이퍼파라미터
  - 0에 가까울수록 : 정규화 효과가 없고
  - 매우 크면 : 가중치가 거의 0에 가까워져 모델이 너무 단순해질 수 있음

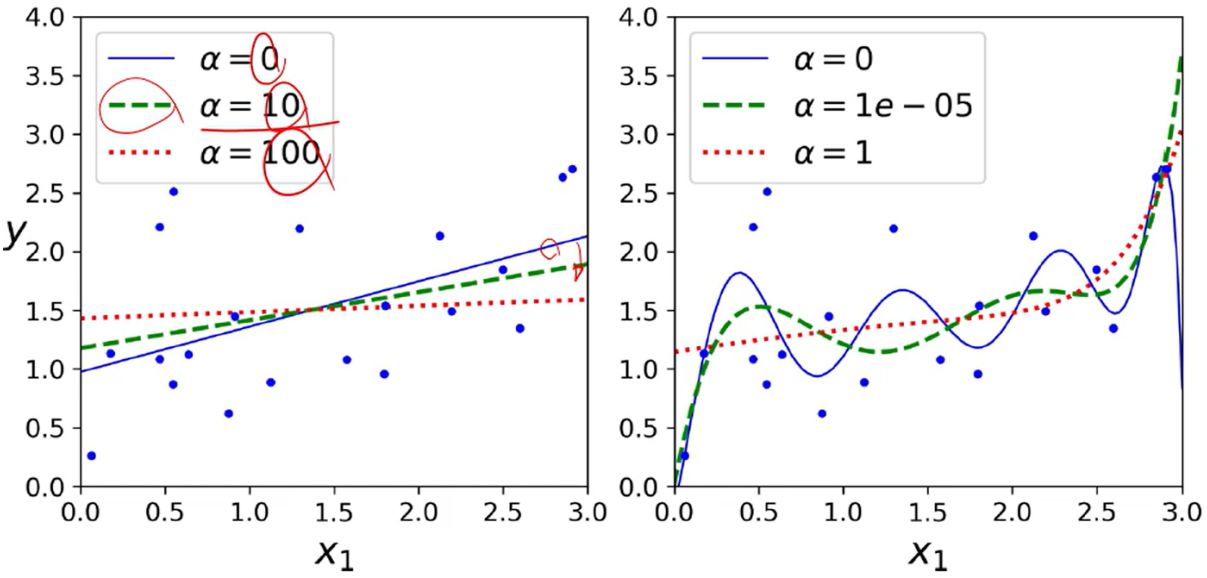


너무 크면 이렇게 flat해짐

- Cost Fucntion : Bias term theta_0만 not regularized

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

- StandardScaler 같은 tool을 가지고 데이터를 scale하는게 좋음
  - (Ridge) Regression은 입력 feature의 scale에 민감하기 때문에 하는게 좋
- 왼쪽은 linear model, 오른쪽은 polynomialFeatures(10차원)
  - 오른쪽: StandardScaler로 스케일됨.
  - 알파 값 높아질수록 variance 감소, bias 증가

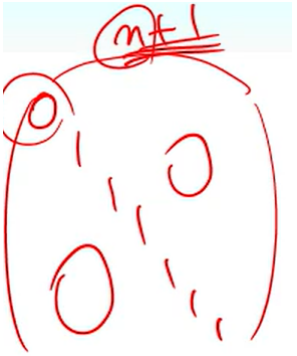


오른쪽 그래프 파란그래프 x가 3 정도 될 때, 급격히 값이 줄어드는걸 보면 overfitting 되어있다는걸 알 수 있음 → regularization 필요함.

- ***Closed form*** : 알파 * A가 추가

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

  - A : (n+1)(n+1) 크기의 identity matrix(항등 행렬 = 단위 행렬)
    - identity matrix: 주 대각선(diagonal) 상의 모든 원소가 1이고, 나머지 원소는 모두 0인 정사각 행렬



왼쪽 위만 0으로하고 나머지 대각선은 1 (→ *A*를 identity matrix로 사용하면 수식이 간결해짐)

- SGD에서는 penalty를 l2로 지정해서 ridge regularization된 linear model로 사용.

```
>>> sgd_reg = SGDRegressor(penalty="l2")
>>> sgd_reg.fit(X, y.ravel())
>>> sgd_reg.predict([[1.5]])
array([1.47012588])
```

- l2 norm을 cost function에 더해준다는 뜻

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

l2 norm

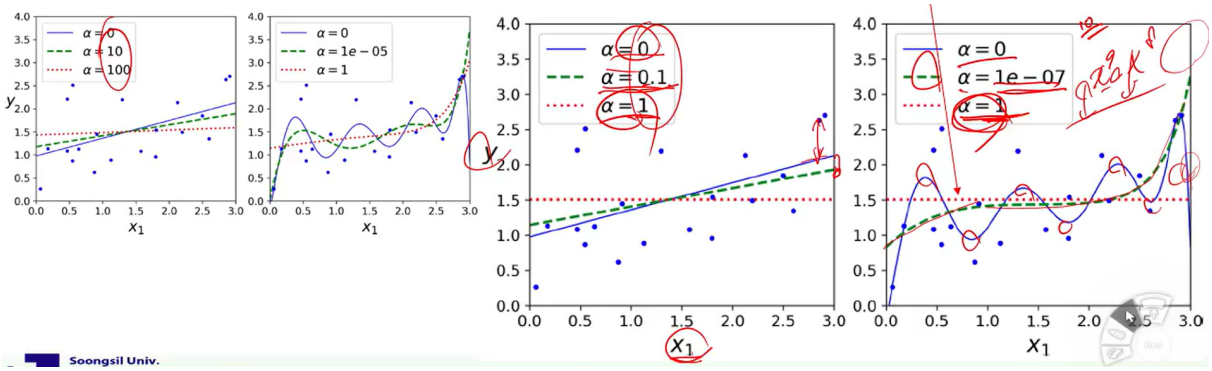
2. Lasso Regression (=L1 Regularization)

(Least Absolute Shrinkage and Selection Operator Regression)

- L1 regularization

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

- 별로 안중요한 feature의 가중치를 제거 (0으로)
- sparse model (feature 가중치가 거의 0인)
  - ex. 세타 1~100 있으면, 10개만 중요한거고 나머지는 0



왼쪽 두개는 ridge regression이고, 오른쪽 두개는 lasso regression. lasso는 ridge와 달리 알파값을 1로만 해도 훨씬 더 강력한 regularization이 적용됨.

3. Elastic Net

- Ridge Regression과 Lasso Regression 중간 성격

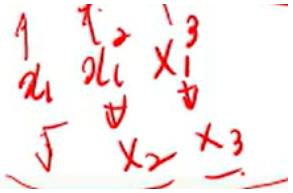
$$J(\theta) = \mathrm{MSE}(\theta) + r\alpha\sum_{i=1}^{n}|\theta_i| + \frac{1-r}{2}\alpha\sum_{i=1}^{n}\theta_i^2$$

- r : mix ratio
  - 0 : 추가된 두 항 중 왼쪽게 0이므로 Ridge regression이 됨
  - 1 : lasso regression이 됨
- 코드에서 l1_ratio가 r임

```
>>> from sklearn.linear_model import ElasticNet
>>> elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
>>> elastic_net.fit(X, y)
>>> elastic_net.predict([[1.5]])
array([1.54333232])
```

+. Regularization 정리

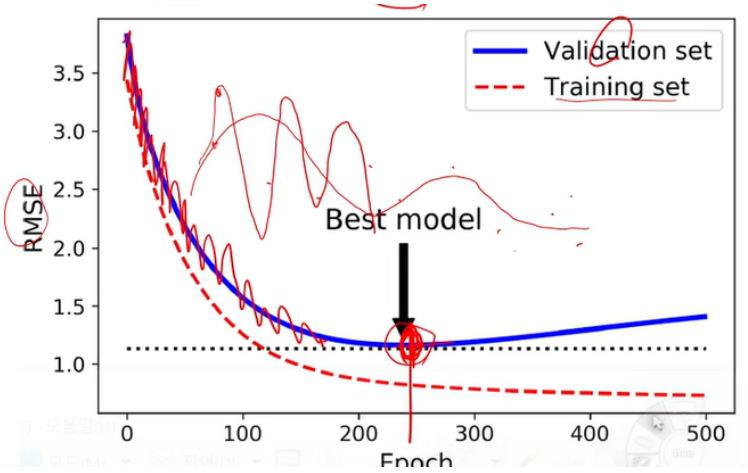
- preferable (거의 대부분의 상황에서 사용하는게 좋음)
- ridge가 일반적으로 좋음 (default)
- 특정한 feature가 중요하면, lasso나 elastic net이 나음.
- 일반적으로는 Elstic Net이 좋음
  - lasso는 입력 feature가 training feature보다 많으면 결과가 이상하게 나오는 경우가 있음
  - 아니면 몇가지 feature가 correlate 강력하게 되어있는 경우



6. Early Stopping

Overfitting을 막는 또 다른 방법

- Gradient Descent처럼 iterative learning algorithms을 regularization
- validation error가 최소가 될 때 학습을 멈춤
- Ex. BGD, high degree polynomial regression 사용하는 경우
  - validation error가 최소가 되면 멈춤
  - 멈췄을 때의 epoch에서의 파라미터를 저장하고, 그걸 내가 사용할 모델로 최종 결정함.
  - training data에만 fit하게 학습되면 general data에 대비가 안됨 → 그래서 어느정도 되었을 때 학습을 멈춰서 overfitting 막음
  - BGD는 validation error 그래프가 비교적 smooth하게 변하지만, SGD나 Mini BGD는 아님
    - 최솟값을 찾기 어려움
    - 그래서 이런 거에 대비가 필요.
      - 5-10개 포인트에서의 평균값을 계산… 등의 방법



> ***Normalization(정규화)***는 데이터를 특정 기준에 맞추어 변환하거나 스케일을 조정하는 과정 (regularization과 다른거임. 한국어 뜻 이름만 똑같음)

```
split the quadratic dataset
X_train, y_train = X[: m // 2], y[: m // 2, 0]
```

```
X_valid, y_valid = X[m // 2 :], y[m // 2 :, 0]

# PolynomialFeatures는 다항식 특성을 추가하여, X_train과 X_valid 각
# x^90 차원, bias는 포함시키지 않는다
# StandardScaler는 표준화(standardization)를 수행하여 각 특성의 평균
preprocessing = make_pipeline(PolynomialFeatures(degree=90, i
                              StandardScaler())

X_train_prep = preprocessing.fit_transform(X_train) # fit and
X_valid_prep = preprocessing.transform(X_valid) # transform

# penalty=None (regularization 하지 않음)
sgd_reg = SGDRegressor(penalty=None, eta0=0.002, random_state
n_epochs = 500
best_valid_rmse = float('inf')
train_errors, val_errors = [], []  # extra code – it's for th

for epoch in range(n_epochs):
    # warm start(직전에 학습되었던 곳에서부터 시작) <-> cold start(
    sgd_reg.partial_fit(X_train_prep, y_train) # single round
    y_valid_predict = sgd_reg.predict(X_valid_prep) # validat
    val_error = mean_squared_error(y_valid, y_valid_predict,
    if val_error < best_valid_rmse: # 최적의 모델이면 (cost funct
        best_valid_rmse = val_error
        best_model = deepcopy(sgd_reg) # copy hyperparameters
                # validation error가 작아질때만 실행되는 위의 두 줄
                # 실제로는 찾으면 빠져나오게 설계함.
```

7. Logistic (Logit) Regression

- 특정한 인스턴스가 특정한 클래스에 속할 확률을 추정하는 것.
- ***로지스틱 회귀(Logistic Regression)는 선형 회귀(Linear Regression)의 결과를 시그모이드 함수(Sigmoid Function) (=로짓 함수(Logit Function))에 넣어서 0 또는 1의 이진 결과를 얻는 모델***
- binary classifier: 이진 분류(Binary Classification) 문제를 해결하기 위해 주로 사용

- 50%보다 크면 속한다. (positive class에 속한다, “1” 출력)
- 아니면, 속하지 않는다. (negative class에 속한다, labeled “0”)

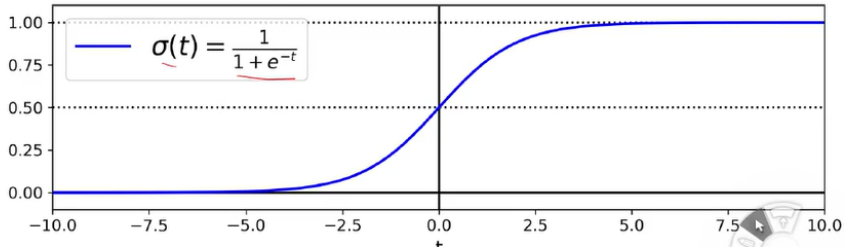
- Estimating Probabilities (어떻게 확률 추정?)
  - 입력 feature의 weighted sum을 구하고, bias term을 더하면 → linear regression의 출력값(x^T*θ = t)
  - 그 출력값이 sigmoid의 입력값이 되어 확률을 구함

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

  - logistic (noted σ(·)) function: sigmoid function (로지스틱 함수는 시그모이드 함수)

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

▶Logit: score t
logit(p)=log(p/(1-p))
p=σ(t): [0, 1]
→ t : (-∞, ∞)



- Traning & Cost Fuction
  - 학습 목적: 파라미터 벡터 θ 구하기
    - model estimates
      - high probabilities for positive instances (y = 1)
      - low probabilities for negative instances (y = 0)
  - Cost function of a Single Training Instance

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

-log(p) very large if p → 0
→ 0 if p → 1 for a positive instance
-log(1-p) very large if p → 1
→ 0 if p → 0 for a negative instance

> - Loss 함수 : 모델 학습 중에 사용되며 모델을 학습하는 데 도움을 주는 함수

> - Cost 함수 : 학습 이후에 모델의 성능을 평가하고 비교하는 데 사용되는 함수

- Cost function over the Whole Training Set : *the average cost*
  - ***logistic regression cost function (= log loss)***

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

왼쪽은 postive, 오른쪽은 negative

- linear regression과 달리 cost function을 최소화하는 θ를 얻는 closed-form equation이 없음
  - gradient descent 같은 iterative 방법을 사용할 수밖에.
- cost function은 U형태. (=convex) → global minimum 찾는게 보장됨
  - lr이 너무 길지 않고, 충분한 시간동안 iteration을 진행한다면!

Decision Boundaries

- 붓꽃 데이터 (Iris dataset): 3가지 종류 (Setosa, Versicolor, Virginica)
  - feature: petal(위쪽 꽃잎) length와 width, sepal(아래쪽 꽃잎) length와 width (cm 단위)
  - 150개의 데이터셋 (많지 않음)

```
from sklearn.datasets import load_iris

iris = load_iris(as_frame=True)
list(iris) # 데이터 파악을 위해 key들을 뽑아봄.
```

✓ 1.3s

```
['data',
 'target',
 'frame',
 'target_names',
 'DESCR',
 'feature_names',
 'filename',
 'data_module']
```

```
iris.data.head(3) # 첫 번째 3개 인스턴스를 출력해봄
```
✓ 0.0s

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2

인스턴스 총 150개 있음

```
iris.target.head(3)  # note that the instances are not shuffled
# 모두 0으로, setosa임.
```
✓ 0.0s

```
0    0
1    0
2    0
Name: target, dtype: int64
```

```
iris.target_names
```
✓ 0.0s

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

0만 보면 뭔지 모르니까, names를 뽑아서 뭔지 알아봄

'U10' 의미: Unicode 10자 이내의 데이터 타입.

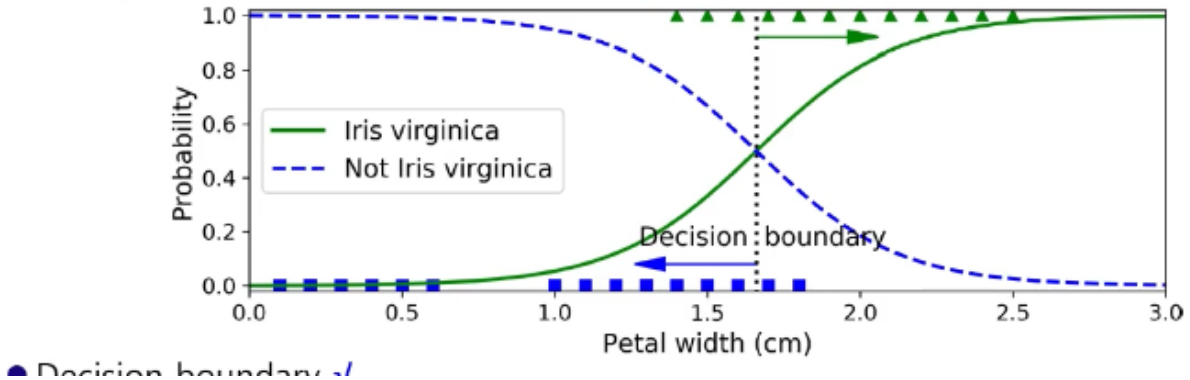
```
# Training
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

X = iris.data[["petal width (cm)"]].values
y = iris.target_names[iris.target] == 'virginica' # virginica이면 1, 아니면 0이 들어감 (binary classification)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train, y_train)
```

feature 3개지만 하나에 대해서 binary classification을 해봄

model's estimated probabilities

1. petal width: 0-3cm

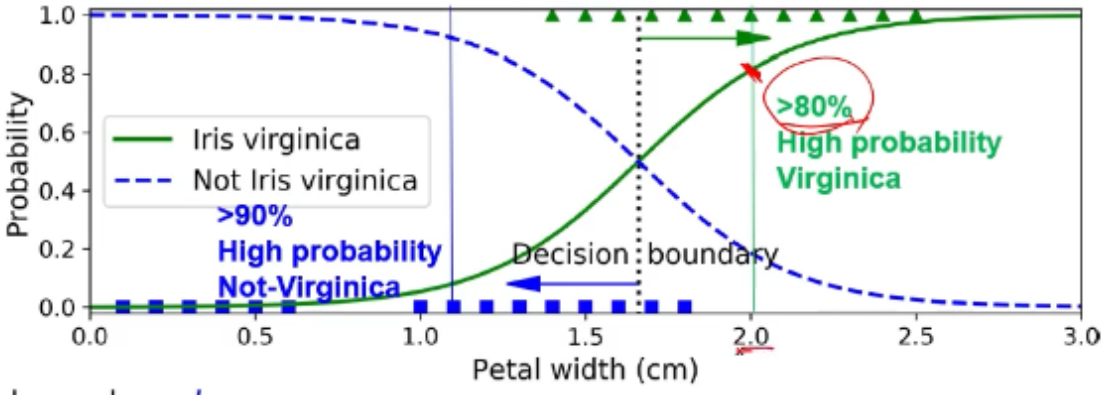
```
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
plt.plot(X_new, y_proba[:, 1], "g-", label="Iris virginica")
plt.plot(X_new, y_proba[:, 0], "b--", label="Not Iris virginica")
# + more Matplotlib code to make the image look pretty
```



중간에 만나는 부분이 확률이 50%인 부분. 이 부분을 기점으로 해서 decision boundary를 정함.

→ decision boundary: 1.6cm @P=0.5

- 초록색 왼쪽 세개는 virginica인데 아니라고, 파란색 오른쪽 두개는 not virginica인데 virginica라고 판명됨.
- plot 해보면 decision boundary 알 수 있음
- decision boundary 이동 시, 입력 인스턴스의 판정 결과가 달라짐. (Tradeoff 상황 발생)



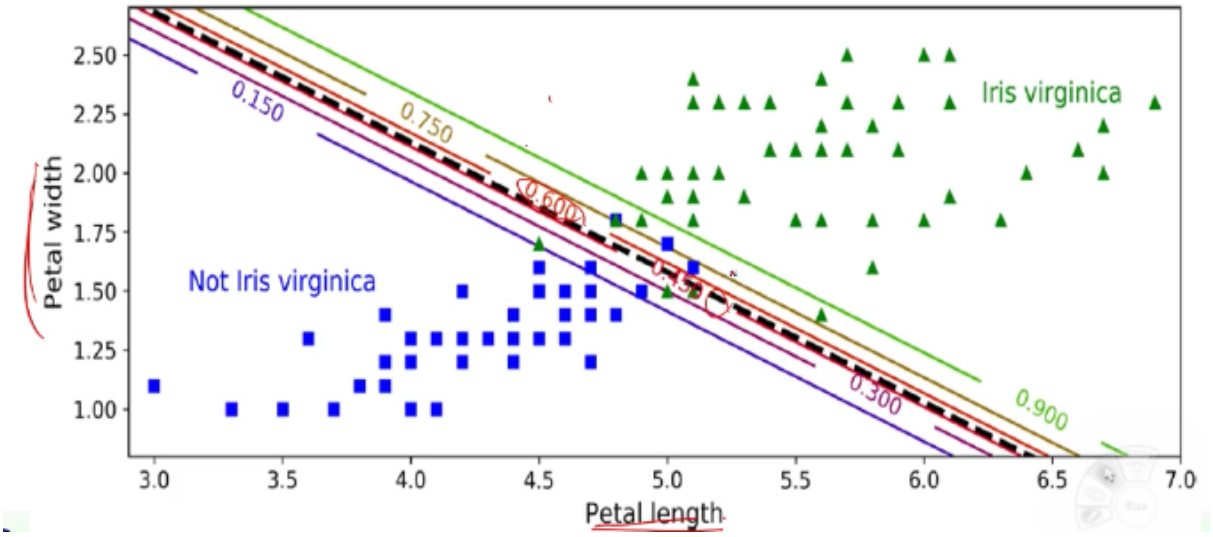
이동된 초록색 세로선: virginica로 판명될 확률 80프로 이상 (not-virginica가 절대 virginica로 판명되지 않음) ↔ 모든 virginica를 virginica로 판명. (파란색 세로선)

→ log_reg.predict([1.7], [1.5]]) : array([1, 0]) //1.7이 들어오면 1(positive), 0이 들어오면 0(negative)로 판정됨

- predict_proba는 예측. predict는 실제 classify된 결과.

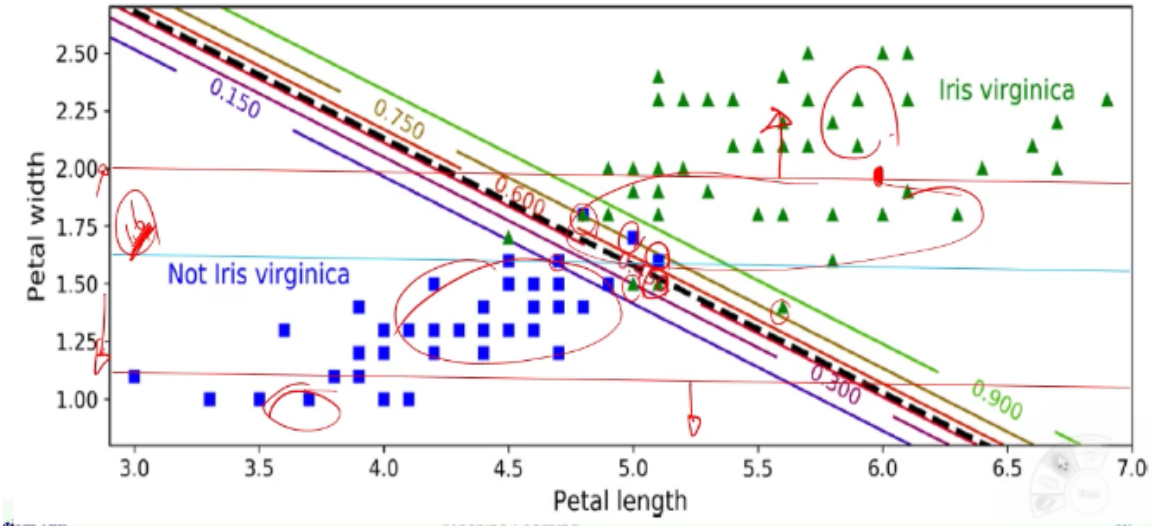
2. 2 features (petal width, length)

- decision boundary: dashed line @P=0.5



가운데 검은색 점선이 decision boundary 0.5인 경우임. 아래위로 움직임에 따라 가운데 부근에 있는 점들의 판정 결과가 달라짐.

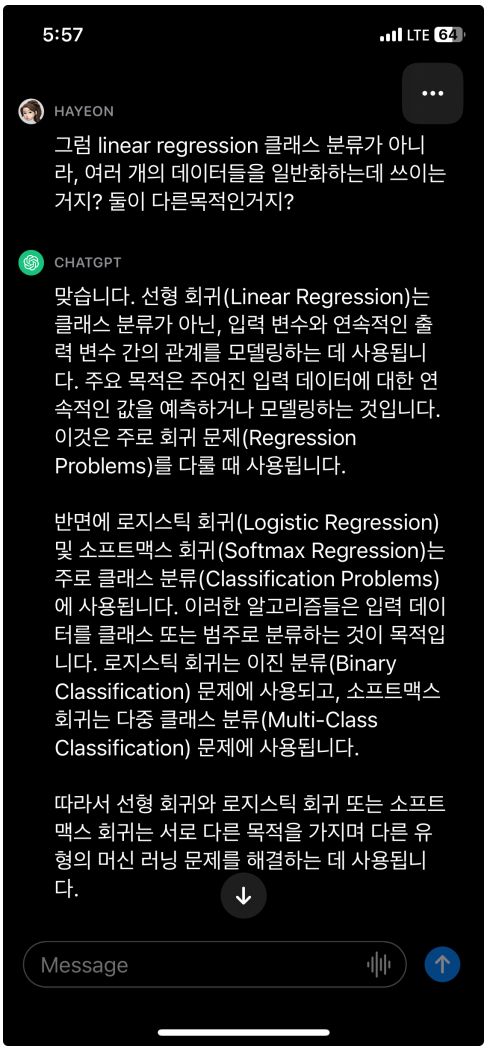
- scimitar-learn에서는 l2 penalty가 default.
- 아래는 input feature를 한 개 했을 때보다 두개 했을 때가 더 오판단이 적은 이유이다.



가로선 - input feature 하나인 경우. 예를 들어 위쪽 빨간석이 decision boundary라고 한다면, 초록색 아래부분이 virginica인데 not virginica라고 판단되는 세모점들이 너무 많음.

- 1개이면, decision boudary를 아래위로 움직였을 때, 잘못 판정되는 샘플의 개수가 너무 많음
- 따라서, 필요한 input feature를 적절히 사용하는게 분류 시 중요하다.

8. Softmax Regression (Multinomial Logistic Regression)



- 다중 클래스를 직접 support하는 일반화된 Logistic Regression.
  - multiple binary classifiers를 합쳐서 학습할 필요 없이, softmax regression으로 해결 가능
    1. 각 클래스 k 별로 s_k(x) 스코어 구함
       - ***Softmax score***

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

2. 각 클래스의 확률 추정 : softmax function을 적용하여 (normalized exponential 모양임)
    - ***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 개수
        - s(x): 스코어 가지고있는 벡터
        - σ(s(x))_k : 인스턴스 x가 class k에 속할 확률
        - p값을 모두 더하면 1이 되겠지 (각각 클래스에 속할 확률들의 합이니까.)

$$\sum \hat{p}_k = 1$$

- *Softmax Regression* ***Classifier Prediction***
    - 클래스 별 p 중에 가장 높은 확률의 클래스에 속한다고 판단하면 됨.

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

    - k를 리턴하는 argmax
- should be used only with mutually exclusive classes (입력이 클래스 1과 5에 동시에 속할 수 있는 건 안됨. / ex.강아지 종이 섞인…)

- Cross Entropy (cost function)
    - 학습 목적: target class에서 높은 확률이 추정되는것 (= cross entropy값 최소화)
        - 한 batch에 대해 모든 instance의 평균을 내는 것, 그게 cost function

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

        - i번째 인스턴스가 class k이면

$$y_k^{(i)} = 1$$

- i번째 인스턴스가 class k가 아니면

$$y_k^{(i)} = 0$$

- *Cross entropy Gradient Vector for class k*

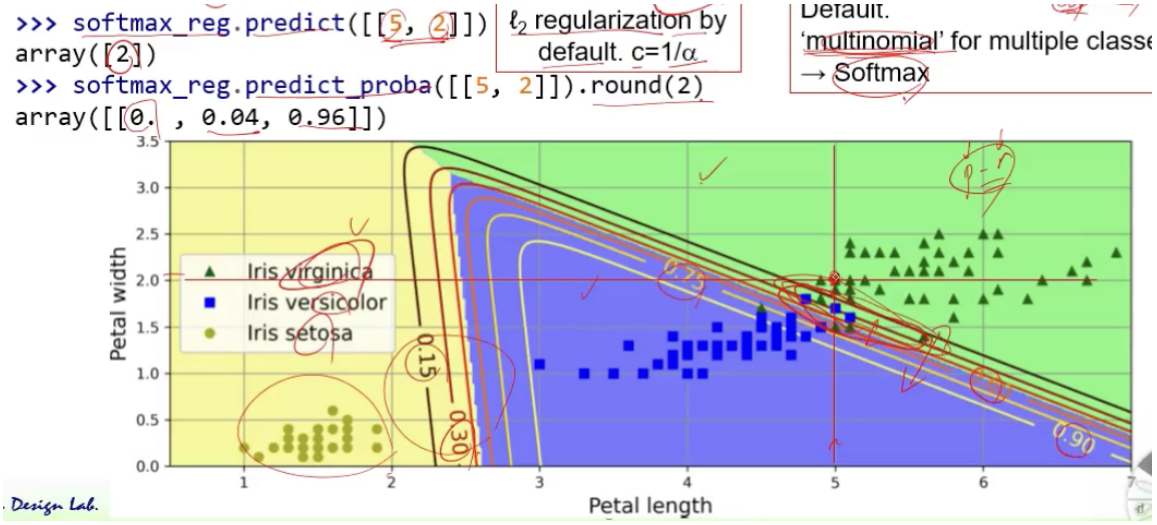
$$\nabla_{\theta^{(k)}} J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{p}_k^{(i)} - y_k^{(i)} \right) \mathbf{x}^{(i)}$$

- Gradient Descent(혹은 다른 최적화 알고리즘)을 사용하여, cost function을 최소화하는 파라미터 행렬 Θ를 구할 수 있음
  - 참고
    - linear regression → (R)MSE, MAE
    - logistic → log loss
    - softmax → cross entropy (classification)

> "Loss Function"는 개별 데이터 포인트에 대한 모델의 오차를 측정하는 함수이며, "Cost Function"는 전체 훈련 데이터 셋에 대한 손실의 평균 또는 합계를 나타내는 함수입니다. 비용 함수는 모델의 학습 과정에서 최적화의 대상이 되며, 모델이 얼마나 잘 작동하는지를 측정하기 위한 지표 중 하나

```
X = iris.data[["petal length (cm)", "petal width (cm)"]].values
y = iris["target"]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

# 만약 binary class이면 ovr, multi class이면 auto
# 강제로 ovr로 설정하면, multi일 때도 ovr 이용해서 분류.
# 지정 안해주면 자동으로 지정함.

# c = 1/alpha (c값 커질수록 l2 정규화 더 많이)
softmax_reg = LogisticRegression(C=30, random_state=42, multi_class='auto')
softmax_reg.fit(X_train, y_train)
```

✓ 0.2s



predict함수로 어떤 클래스인지 판별 → predict_proba로 어떤 확률로 판정되었는지 확인.

4장 Summary

- linear regression: 1차 함수로 egression하기.
  - polynomial regression은 polynomial feature로 쉽게 할 수 있음
- 학습의 목적: bias, weight라는 모델 파라미터를 찾기 위함임.
- 최적의 모델 파라미터 찾으려면, cost function or loss function을 최소화시키는 파라미터를 찾으면 됨.
  - 엄밀히 말하면 다르지만, 꼭 그럴필요 없음
  - Linear regression에서는 MSE/RMSE 사용. (score라고도 불림. score 값이 클수록 안좋은 결과임.)
- BGD: 모든 학습 데이터에 대해 예측하고 업데이트
  - SGD: 하나의 인스턴스마다 업데이트
  - Mini Batch: 일정한 개수의 subset으로 업데이트 함(개수 설정 가능, 요즘 거의 이거 씀)
- 학습이 제대로 안되면 overfitting, underfitting - learning curve로 확인 가능
  - 모델 바꾸던지… 다시 학습해야
- 하이퍼파라미터/ learning rate : 학습 모델 구조, 학습에 영향을 미치는 파라미터 (vs. 그냥 파라미터는 모델에 영향을 줌)

- 학습이 제대로 진행되지 않은걸 확인하는 것들: Bias, Variance, Irreducible Error (굳이 몰라도 됨)
- Overfitting 억제 방법 : 1) regularization, 2) stopping
- logistic regression: sigmoid function으로 probability 구하여 classification
- decision boundary에 따라 최종 판정 결과 달라질 수 있다.
- softmax regression: logistic regression을 multi classification에 대해 한거