인공지능 실습 Chapter 2

Machine Learning

포항공과대학교 컴퓨터공학과

IRL 정보통신연구소
POSTECH INFORMATION RESEARCH LABORATORIES

POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Chapter 2

- **Supervised Learning (지도 학습)**
  - Sentiment Analysis
    - 감정 분류하기
    - 영화 리뷰 점수 예측하기
- **Unsupervised Learning (비지도 학습)**
  - k-means Clustering
    - 영화 리뷰 클러스터링

Sentiment Analysis

- **Sentiment Analysis** (감정분석) :
  - 주어진 텍스트의 감정을 자동으로 분석하는 작업
  - 감정 분류 : **부정적 (negative)** / **긍정적 (positive)**
  - 감정 예측 : **-5점** ~ **+5점**
- **Rotten Tomatoes**

Feature Extraction

- 텍스트 데이터로부터 feature vector $\boldsymbol{\phi}(\boldsymbol{x})$를 추출하는 작업
- ex) 각각의 단어를 feature로 사용하는 경우,
- ☺ “pretty good” → {‘pretty’ : 1, ‘good’ : 1}
- ☺ “not bad” → {‘not’ : 1, ‘bad’ : 1}
- ☹ “ewww so dirty” → {‘ewww’: 1, ‘so’ : 1, ‘dirty’ : 1}

2.1. Sentiment Classification (감정 분류)

Linear Predictor

- Feature들의 linear weighted sum으로 감정을 분류

$$f_{\mathbf{w}}(x) = sign\big(\mathbf{w} \cdot \phi(x)\big) = \begin{cases} +1 & if\ \mathbf{w} \cdot \phi(x) > 0 \\ -1 & if\ \mathbf{w} \cdot \phi(x) < 0 \end{cases}$$

- weight vector의 의미?
  - $\phi(x_1) = \{`very': 1, `interesting': 1,\ `movie': 1\}$
  - $\phi(x_2) = \{`very': 1, `boring': 1\}$
  - $\boldsymbol{w} = \{`very': ??, `interesting': ??, `boring': ??, `movie': ??\}$

Linear Predictor

- Feature들의 linear weighted sum으로 감정을 분류

$$f_{\mathbf{w}}(x) = sign\big(\mathbf{w} \cdot \phi(x)\big) = \begin{cases} +1 & if\ \mathbf{w} \cdot \phi(x) > 0 \\ -1 & if\ \mathbf{w} \cdot \phi(x) < 0 \end{cases}$$

- weight vector의 의미?
  - $\phi(x_1) = \{`very' : 1, `interesting' : 1,\ `movie' : 1\}$
  - $\phi(x_2) = \{`very' : 1, `boring' : 1\}$
  - $\boldsymbol{w} = \{`\boldsymbol{very}' : 0, `\boldsymbol{interesting}' : 1, `\boldsymbol{boring}' : -1, `\boldsymbol{movie}' : 0\}$

Loss Function

- 감정 분류를 위한 Loss function을 정의
- 아래와 같이 정의되는 **Hinge loss**를 사용

$$\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \max\{0, 1 - \mathbf{w} \cdot \phi(x)y\}$$

  - x : input text
  - $\phi(x)$: feature vector
  - $\boldsymbol{w}$: weight vector
  - y : 감정 label

Stochastic Gradient Descent

- 목표 : 앞서 정의한 Hinge loss를 최소화하는 $\boldsymbol{w}$를 학습
- 방법 : Stochastic Gradient Descent (SGD)

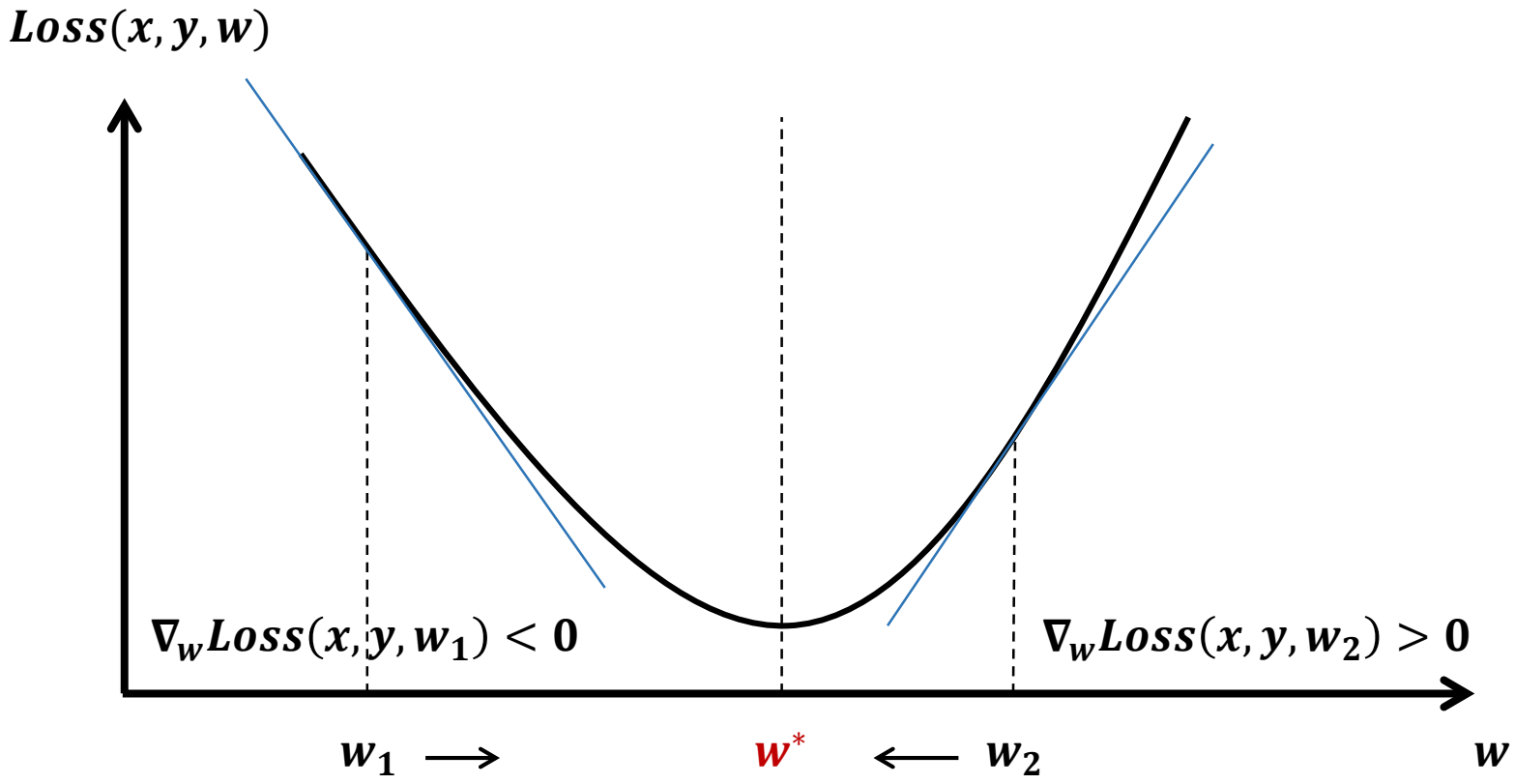
$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathrm{Loss}_{\mathrm{hinge}}(x, y, \mathbf{w})$$

- Step 1 : Loss function의 gradient 식을 구한다.
- Step 2 : 학습데이터를 이용하여 위의 업데이트를 반복한다.

Stochastic Gradient Descent

- Example

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathrm{Loss}(x, y, \mathbf{w})$$

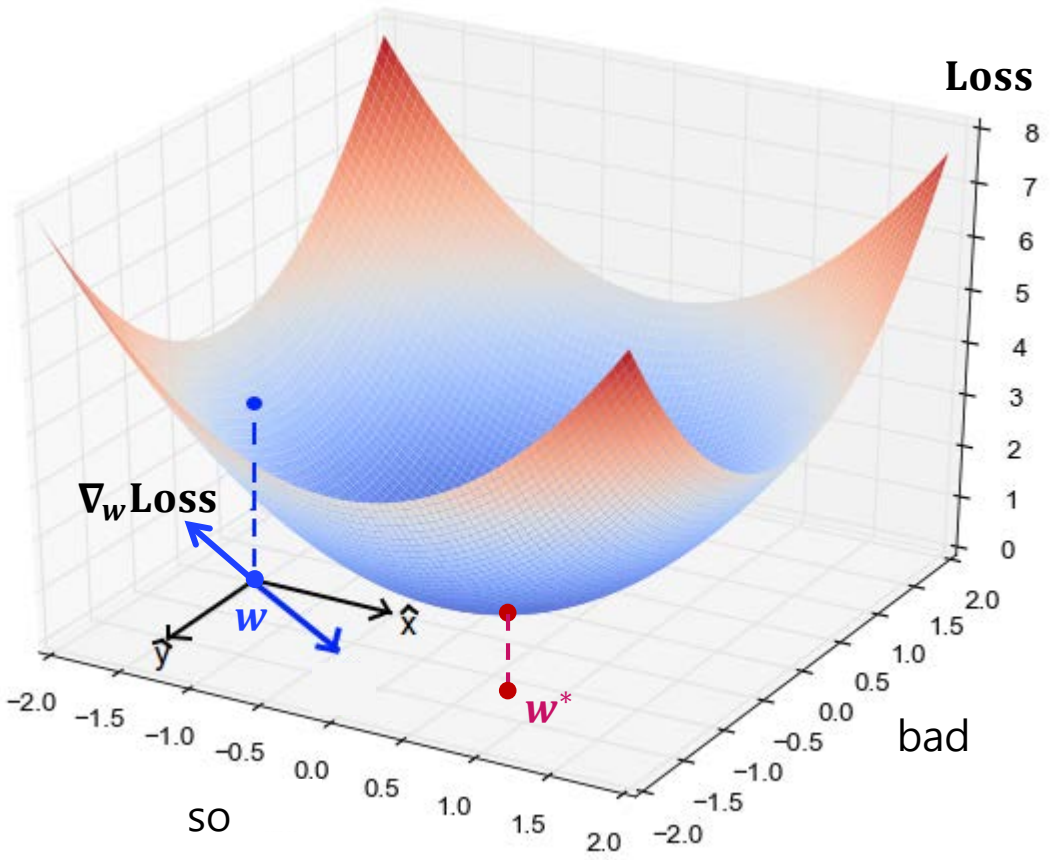


Stochastic Gradient Descent

- Example

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathrm{Loss}(x, y, \mathbf{w})$$

$$\boldsymbol{w} = \{\text{`}so\text{'}: ?, \text{`}bad\text{'}: ?\}$$



A. SGD 연습

- 아래 네 개의 데이터로 SGD를 통해 $\boldsymbol{w}$를 학습해보자.
- 조건: Step size 파라미터 $\eta = 1$, 벡터 초기화 $\boldsymbol{w} = [0, \ldots, 0]$
- ☺ pretty good
- ☹ bad plot
- ☹ not good
- ☺ pretty scenery

B. Feature Extractor 구현

- extractWordFeatures 함수를 구현해보자.
- 기능: 문자열 x를 받아 feature vector $\phi(x)$를 반환한다.

```
def extractWordFeatures(x):
    """
    Extract word features for a string x. Words are delimited by
    whitespace characters only.
    @param string x:
    @return dict: feature vector representation of x.
    Example: "I am what I am" --> {'I': 2, 'am': 2, 'what': 1}
    """
    # BEGIN_YOUR_CODE
    raise Exception("Not implemented yet")
    # END_YOUR_CODE
```

- 조건: feature vector는 dictionary 자료형으로 구현한다.

C. Stochastic Gradient Descent 구현

- learnPredictor 함수를 구현해보자.
- 기능: SGD를 이용하여 $\boldsymbol{w}$를 학습한다.

```
def learnPredictor(trainExamples, testExamples, featureExtractor, numIters, eta):
    '''
```

```
    weights = {}
    # BEGIN_YOUR_CODE
    raise Exception("Not implemented yet")
    # END_YOUR_CODE
    return weights
```

- 조건: 각 iteration이 끝날 때마다 training error와 test error를 출력한다. (util.py에 정의된 evaluatePredictor를 이용하자.)

D. 데이터 생성기 구현

- generateExample 함수를 구현해보자.
- 기능: $\boldsymbol{w}$로 정확하게 분류되는 임의의 샘플을 생성한다.

```
def generateDataset(numExamples, weights):
    '''
    Return a set of examples (phi(x), y) randomly
    which are classified correctly by |weights|.
    '''
    random.seed(42)
    # Return a single example (phi(x), y).
    # phi(x) should be a dict whose keys are a subset of the ke
    # and values can be anything (randomize!) with a nonzero sc
    # y should be 1 or -1 as classified by the weight vector.
    def generateExample():
        # BEGIN_YOUR_CODE
        raise Exception("Not implemented yet")
        # END_YOUR_CODE
        return (phi, y)
    return [generateExample() for _ in range(numExamples)]
```

E. Error case 분석

- Test c-2를 실행시키면 learnPredictor 함수와 polarity.train 파일을 이용하여 감정 분류 모델을 학습하고, 최종적으로 학습된 weights 파일과 에러를 분석한 error-analysis 파일이 생성된다.
- error-analysis 파일을 열어, 잘못 분류된 데이터들을 바탕으로 그 데이터들이 왜 제대로 분류되지 못했는지 이유를 찾아보자.

F. 음절 단위 Feature Extractor 구현

- extractCharacterFeatures 함수를 구현해보자.
- 기능: 문자열 x를 받아 character feature vector를 반환한다.
- Character n-grams: 연속된 n개의 문자를 하나의 feature로 간주하는 모델

```
def extractCharacterFeatures(n):
    '''
    EXAMPLE: (n = 3) "I like tacos" -->
    {'Ili': 1, 'lik': 1, 'ike': 1, ...
    You may assume that n >= 1.
    '''
    def extract(x):
        # BEGIN_YOUR_CODE
        raise Exception("Not implemented yet")
        # END_YOUR_CODE
    return extract
```

G. Character Feature를 이용한 감정 분류

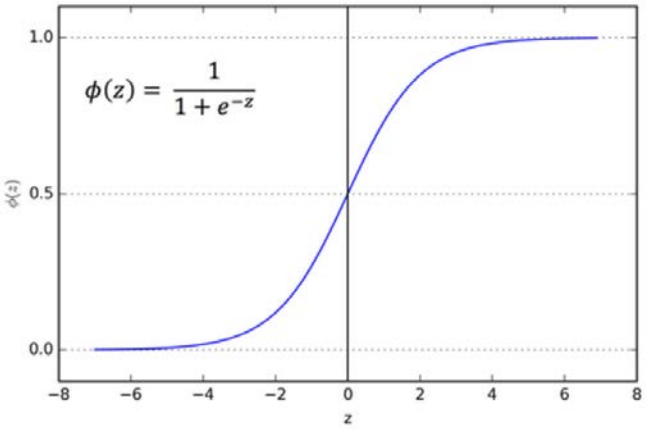
- learnPredictor 함수에서 앞서 구현한 character feature를 사용하도록 변경해보자. n을 바꿔가며 Test c-2를 실행시켜 test error를 가장 작게 만드는 n을 찾아보자.
- Word feature를 사용했을 때보다 Character n-gram feature를 사용했을 때 더욱 정확하게 감정을 분류할 수 있는 하나의 예시를 직접 만들어보고, 그 이유를 설명하라.

2.2. Predicting Moive Rating
(영화 리뷰 점수 예측)

영화 리뷰 점수 예측

- Non-linear 함수인 logistic 함수 $\sigma(z)$를 사용하여 영화 리뷰 x를 input으로 받아, 점수 $\sigma(\boldsymbol{w} \cdot \phi(x))$를 output으로 반환
- **Logistic function**
  - 범위가 없는 실수를 [0, 1] 범위의 값으로 변환
  - z가 커질수록 1에, z가 작아질수록 0에 가까워짐

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



H. Loss function 정의

- Squared loss (에러의 제곱을 반영하는 loss)를 사용하여 이 문제의 Loss funciton을 정의해보자.

I. Loss의 Gradient 계산

- 앞서 정의한 loss function의 gradient를 계산해보자.
- 힌트1 : 답은 예측값인 $p = \sigma(\boldsymbol{w} \cdot \phi(x))$ 를 이용하면 간단하게 나타낼 수 있다.
- 힌트2 : 아래의 logistic function 미분 결과를 활용하자

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

$$\frac{d}{dx}\sigma(x) = \frac{e^x \cdot (1+e^x) - e^x \cdot e^x}{(1+e^x)^2} = \frac{e^x}{(1+e^x)^2} = \sigma(x)(1-\sigma(x))$$

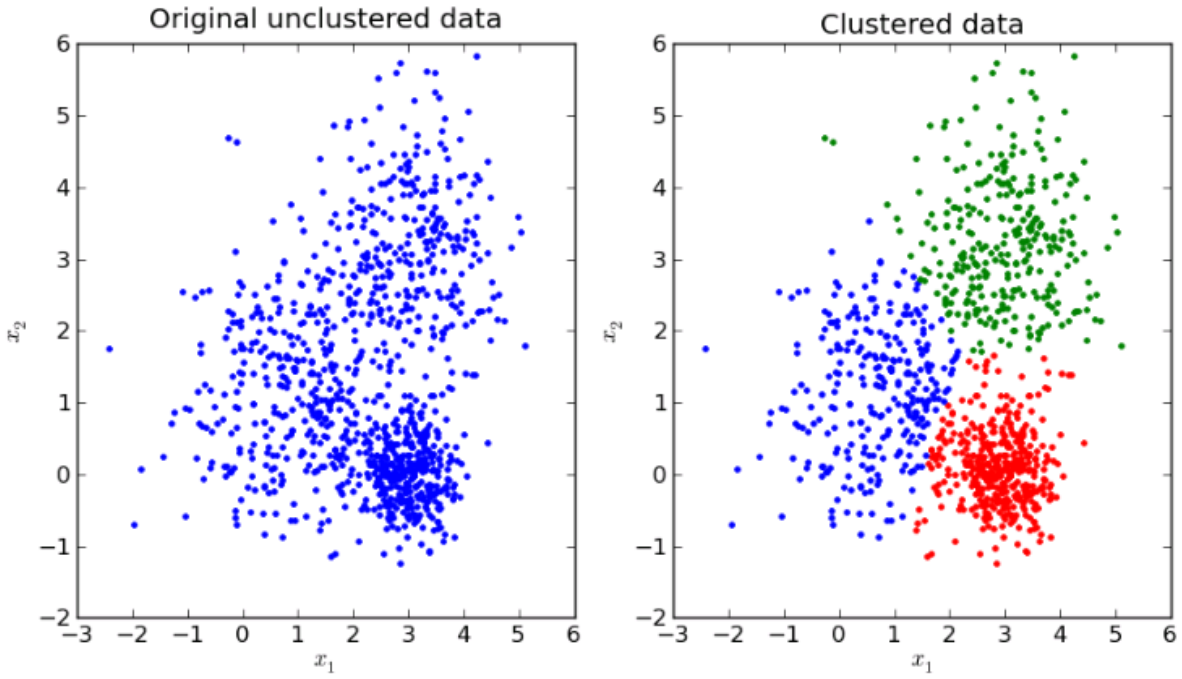
I. Loss의 Gradient 계산

- 유도 과정

2.3. k-means Clustering
(k-means 군집화)

K-means Clustering

- 주어진 데이터를 k개의 cluster로 묶는 알고리즘
- 각 cluster 내 데이터들의 분산이 최소화도록 학습
- 목표: cluster 할당 변수 $\{z_i\}$와 cluster center $\{\mu_j\}$ 계산



K-means Clustering

- Loss function: $\text{Loss}_{\text{kmeans}}(z, \mu) = \sum_{i=1}^{n} \|\phi(x_i) - \mu_{z_i}\|^2$
- **Step 1: Assignment Step**
  - 각각 데이터에 대해 cluster center가 가장 가까운 cluster에 할당

$$z_i \leftarrow \arg\min_{k=1,\ldots,K} \|\phi(x_i) - \mu_k\|^2$$

- **Step 2: Update Step**
  - Cluster에 속하는 데이터들의 평균 값으로 업데이트

$$\mu_k \leftarrow \frac{1}{|\{i : z_i = k\}|} \sum_{i:z_i=k} \phi(x_i)$$

L. k-means Clustering 연습

- $k = 2$ 일때, 아래의 데이터셋에 대해서 k-means clustering 알고리즘을 직접 수행해보자. (결과: 최종적인 cluster assignment 변수 z_i 와 cluster center μ_j)

$$\phi(x_1) = [0, 0],\ \phi(x_2) = [0, 1],\ \phi(x_3) = [2, 0],\ \phi(x_4) = [2, 2]$$

- 초기 cluster center를 아래 두 가지 경우로 설정한 후, 각각의 k-means clustering의 결과를 비교해보자.
  - Case 1: $\mu_1 = [-1, 0]$ and $\mu_2 = [3, 2]$
  - Case 2: $\mu_1 = [1, -1]$ and $\mu_2 = [0, 2]$

M. k-means Clustering 구현

- kmeans 함수를 구현해보자.

```
def kmeans(examples, K, maxIters):
    '''
    Return: (length K list of cluster centroids,
            list of assignments,
            final reconstruction loss)
    '''
    # BEGIN_YOUR_CODE
    raise Exception("Not implemented yet")
    # END_YOUR_CODE
```

- 함수의 요구사항에 맞추어서 반환하는 코드도 작성한다.
- 조건: k개의 cluster center는 랜덤하게 선택된 k개의 training sample로 초기화한다.