

# Logistic and Lasso Regression (simple document classification)

Hyunjoong Kim

[soy.lovit@gmail.com](mailto:soy.lovit@gmail.com)

[github.com/lovit/{soynlp,soykeyword}](https://github.com/lovit/{soynlp,soykeyword})

# Logistic Regression

---

- Logistic Regression (LR)은 대표적인 binary classification 알고리즘
  - positive class에 속할 점수를  $[0, 1]$  사이로 표현하기 때문에 확률 모형처럼 이용

$$y_{\theta}(x) = \frac{1}{1 + \exp(-\theta^T x)}$$

- 학습 데이터  $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$  가 주어졌을 때, loss function은

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

# Logistic Regression

---

- Softmax regression은 Logistic regression의 multi class classification 버전

$$h_{\theta}(x) = \begin{bmatrix} P(y = 1|x; \theta) \\ \vdots \\ P(y = K|x; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^K \exp(\theta^{(j)T} x)} \exp \left( \begin{bmatrix} \theta^{(1)T} x \\ \vdots \\ \theta^{(K)T} x \end{bmatrix} \right)$$

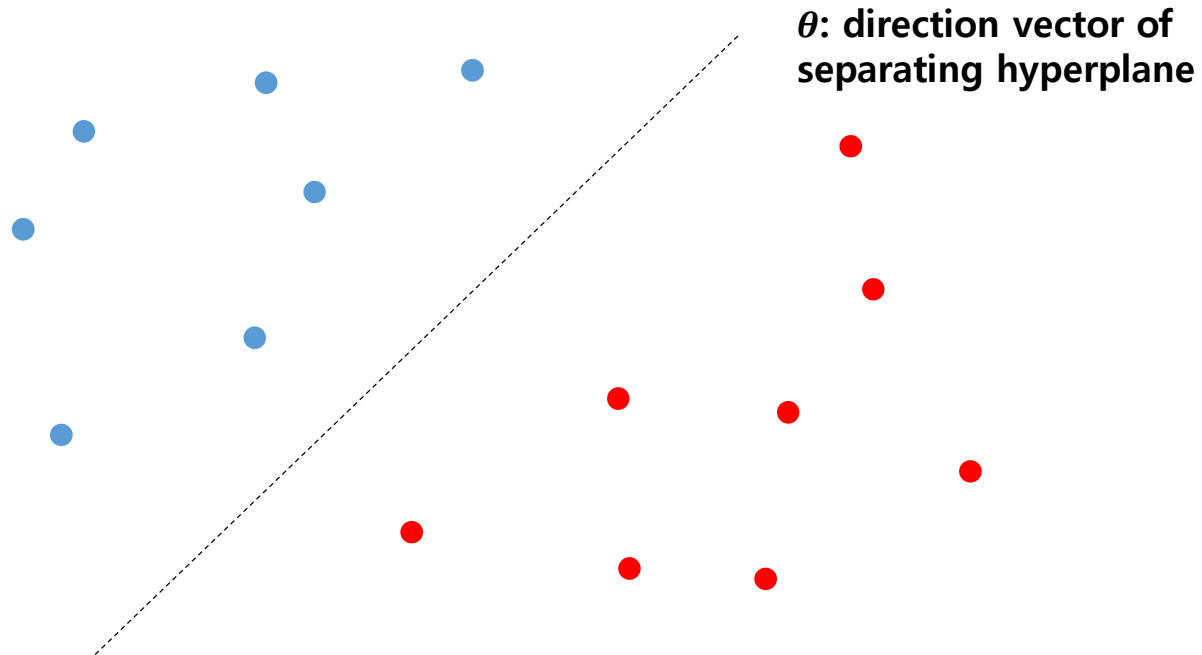
- 학습 데이터  $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$  가 주어졌을 때, loss function은

$$J(\theta) = - \left[ \sum_{i=1}^m \sum_{k=1}^K 1\{y^{(i)} = k\} \log \frac{\exp(\theta^{(k)T} x^{(i)})}{\sum_{j=1}^K \exp(\theta^{(j)T} x^{(i)})} \right]$$

# Logistic Regression

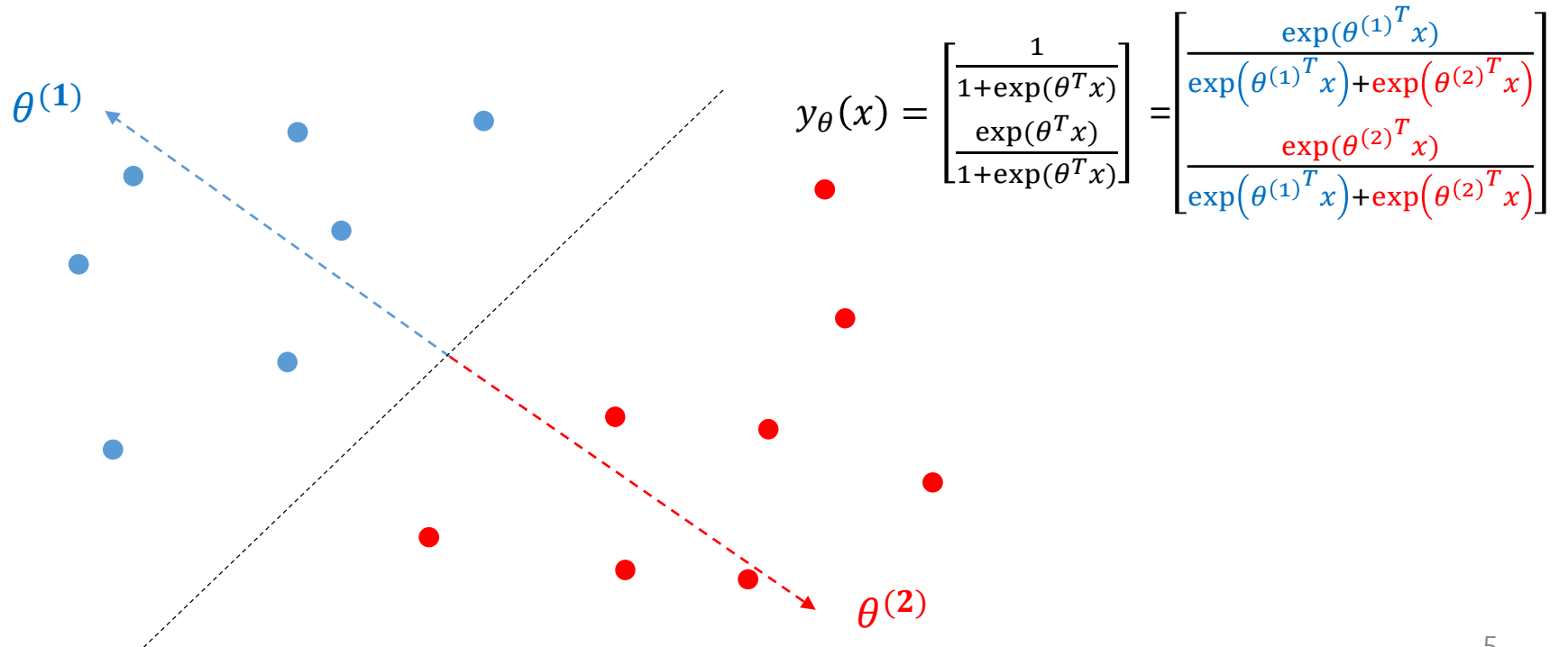
- 일반적으로 LR은 두 클래스의 경계면을 학습한다고 표현합니다

$$y_{\theta}(x) = \frac{1}{1 + \exp(\theta^T x)}$$



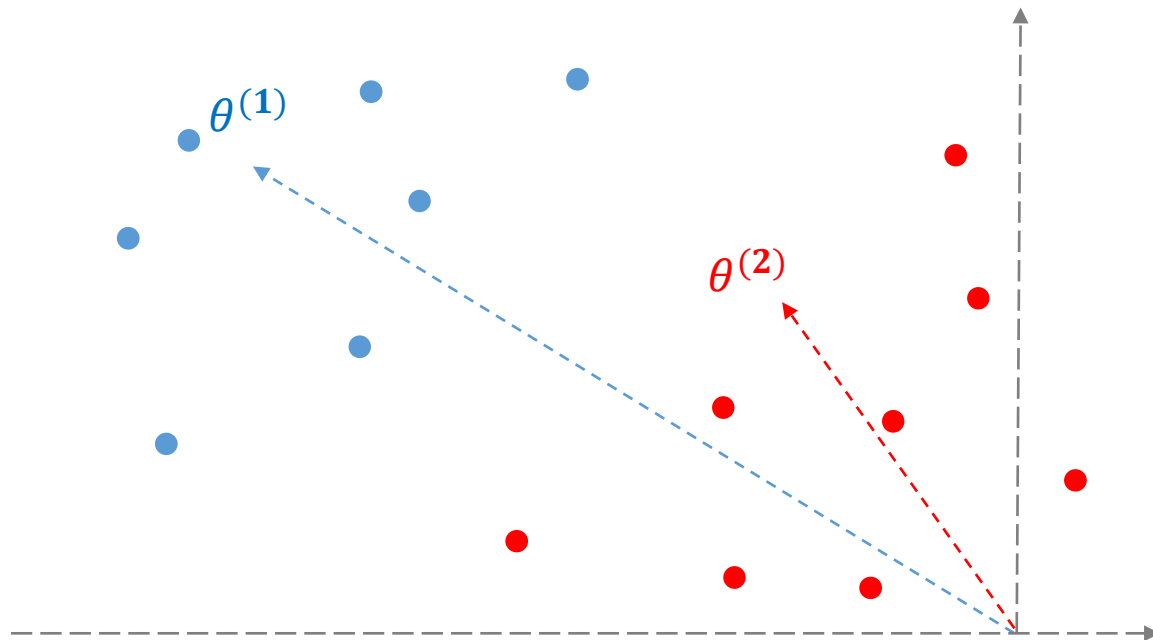
# Logistic Regression

- Softmax regression 표현하면  $\theta = \theta^{(2)} - \theta^{(1)}$ 이며,  $\theta^{(i)}$ 는 클래스  $i$ 의 대표 벡터로 해석할 수 있습니다
- Cosine measure 처럼  $\theta^{(i)}$ 의 방향성이 중요합니다



# Logistic Regression

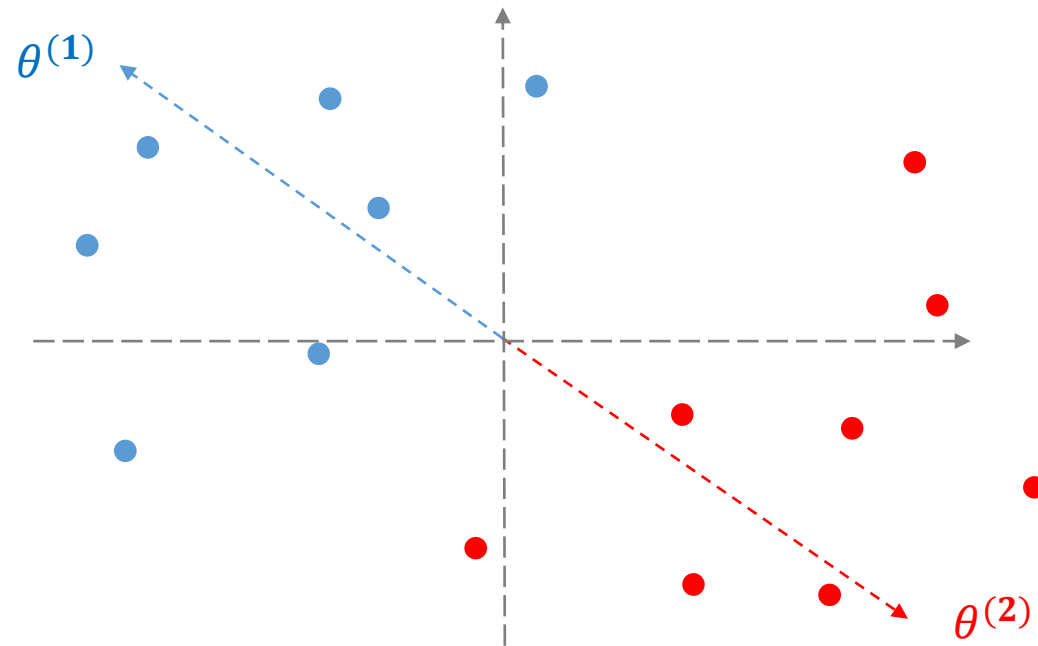
- 두 클래스의 데이터가 같은 방향에 있으면  $\theta^{(i)}$  만으로는 두 클래스를 구분하기 어려울 수도 있습니다
  - 예를 들어, 단어 빈도 벡터의 값은 모두 양수입니다



# Logistic Regression

- Bias term은 클래스를 잘 구분하도록  $x$ 를 평행이동 시킵니다

$$\exp(\theta^T x) = \exp(\theta_0 + \theta_1 x_1 + \dots + \theta_p x_p) = \exp(\theta_1(x_1 - k_1) + \dots + \theta_p(x_p - k_p))$$



# Logistic Regression

- Coefficient  $\theta_{kj}$ 는 문서 종류  $k$ 에 대한 단어  $j$ 의 가중치 (혹은 점수) 입니다

$y$  = '연애뉴스'에 대한 대표벡터 계수

$\theta(\text{연애뉴스}) =$

{
무대: 1.3,
공연: 2.2,
가수: 1.5,
예능: 2.3,
...
외교: -2.2,
정책: -3.3,
무역: -2.5,
...
보였다: 0.01,
이었다: -0.01
}

$x$  = '연애뉴스'의 term frequency vector

$x =$

{
무대: 5,
공연: 3,
가수: 2,
예능: 5,
...
외교: 0,
정책: 0,
무역: 0,
...
보였다: 5,
이었다: 9,
}



# Logistic Regression

- Coefficient  $\theta_{kj}$ 는 문서 종류  $k$ 에 대한 단어  $j$ 의 가중치 (혹은 점수) 입니다

$y$  = '연애뉴스'에 대한 대표벡터 계수

{	
	{
	무대: 1.3,
	공연: 2.2,
	가수: 1.5,
	예능: 2.3,
$\theta(\text{연애뉴스}) \Rightarrow$	...
	외교: -2.2,
	정책: -3.3,
	무역: -2.5,
	...
	보였다: 0.01,
	이었다: -0.01
}	

'연애뉴스' 카테고리에  
높은 점수를 주는 단어들

$x$  = '연애뉴스'의 term frequency vector

{	
	무대: 5,
	공연: 3,
	가수: 2,
	예능: 5,
$x =$	...
	외교: 0,
	정책: 0,
	무역: 0,
	...
	보였다: 5,
	이었다: 9,
}	

# Logistic Regression

- Coefficient  $\theta_{kj}$ 는 문서 종류  $k$ 에 대한 단어  $j$ 의 가중치 (혹은 점수) 입니다

$y =$  '연애뉴스'에 대한 대표벡터 계수

$\theta(\text{연애뉴스}) =$

{

무대: 1.3,  
공연: 2.2,  
가수: 1.5,  
예능: 2.3,  
...  
외교: -2.2,  
정책: -3.3,  
무역: -2.5,  
...  
보였다: 0.01,  
이었다: -0.01

}

문서 판별에 영향이  
없는 단어들

$x =$  '연애뉴스'의 term frequency vector

$x =$

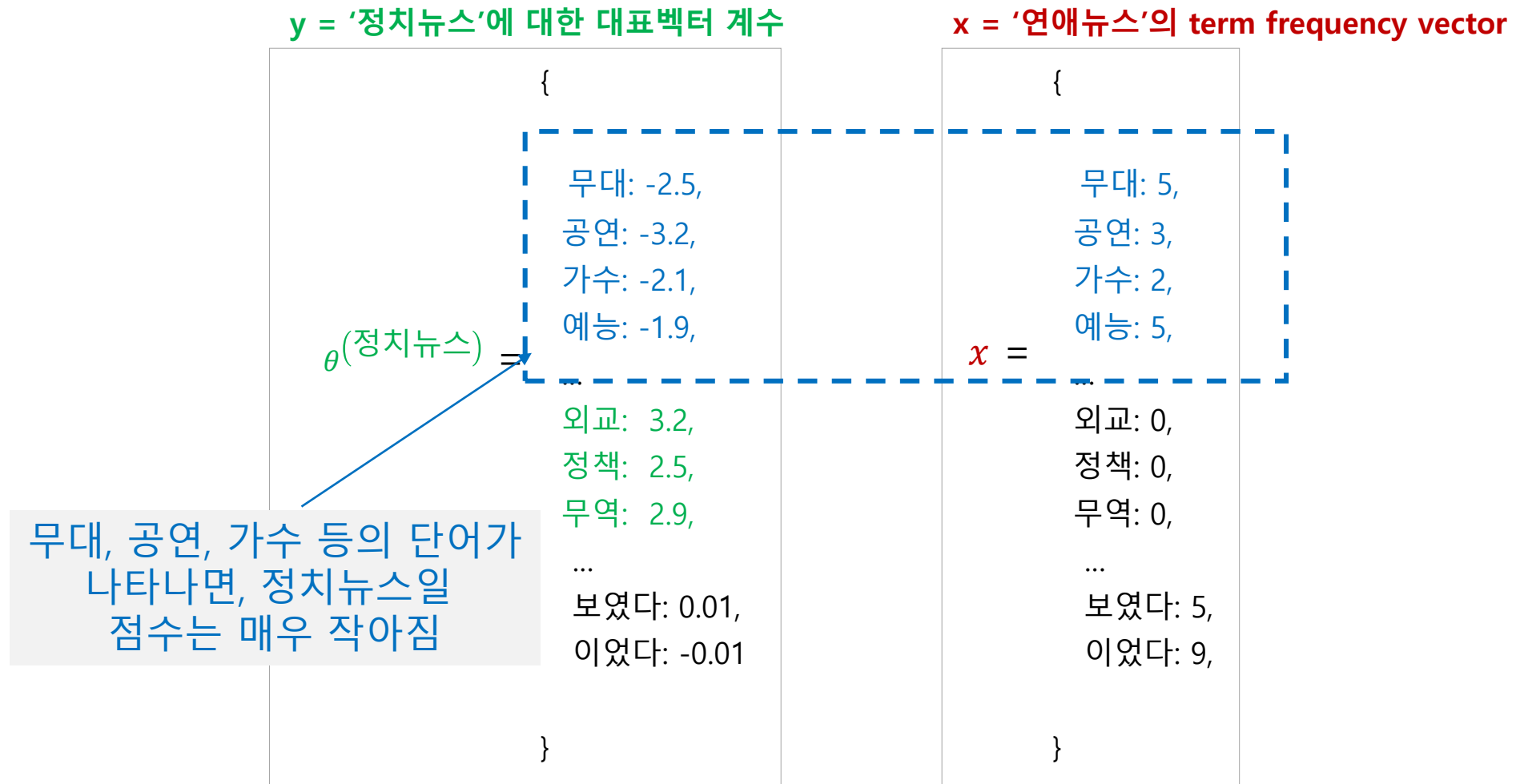
{

무대: 5,  
공연: 3,  
가수: 2,  
예능: 5,  
...  
외교: 0,  
정책: 0,  
무역: 0,  
...  
보였다: 5,  
이었다: 9,

}

# Logistic Regression

- Coefficient  $\theta_{kj}$ 는 문서 종류  $k$ 에 대한 단어  $j$ 의 가중치 (혹은 점수) 입니다



# L2 Logistic Regression

---

- scikit-learn의 Logistic Regression 모델

```
from sklearn.linear import LogisticRegression
```

```
model = LogisticRegression(C=1, penalty='l2')
```

```
model.fit(train_x, train_y)
```

- penalty default = l2
- $C = \frac{1}{\lambda}$ , C가 클수록 regularization은 적게 됩니다.
- train\_x: numpy.ndarray, scipy.sparse
- train\_y: train\_x와 row 길이가 같은 list-like of str/int

# Coefficients of logistic model

---

- Coefficient는 해당 클래스의 가중치와 같습니다

```
coefficients = model.coef_.tolist()  
coefficients[0][:5]
```

```
[-0.555031, 0.0, 0.0, 0.0, 0.0]
```

- coefficients[j]는 j 번째 클래스의 가중치입니다
- Binary classifier이기 때문에 한 개의 클래스 값만 존재합니다
  - positive class에 대한 가중치가 됩니다

# Regularization

---

- p-norm은 벡터의 크기를 정의하는 방법입니다

- $|X|_p = \sqrt[p]{|X_1|^p + \dots + |X_q|^p}$  로 정의되며,  $X_j$ 는 j번째 차원의 값

- $p = 2$ 이면, L2 norm (Euclidean distance)

- $|(3, 0, 4)|_{p=2} = \sqrt{|3|^2 + 0^2 + |4|^2} = 5$

# Regularization

---

- p-norm은 벡터의 크기를 정의하는 방법입니다

- $|X|_p = \sqrt[p]{|X_1|^p + \dots + |X_q|^p}$  로 정의되며,  $X_j$ 는 j번째 차원의 값

- $p = 1$  이면, X의 각 차원의 값의 절대값의 합. (Manhattan distance)

- $|(3, 0, -4)|_{p=1} = \sqrt[1]{|3|^1 + |0|^1 + |-4|^1} = 7$

# Regularization

---

- p-norm은 벡터의 크기를 정의하는 방법입니다

- $|X|_p = \sqrt[p]{|X_1|^p + \dots + |X_q|^p}$  로 정의되며,  $X_j$ 는 j번째 차원의 값

- $p = 0$  이면,  $X$  에서 0이 아닌 차원의 갯수

- $|(3, 0, 4)|_{p=0} = |3|^0 + 0^0 + |4|^0 = 2$



# Regularization

---

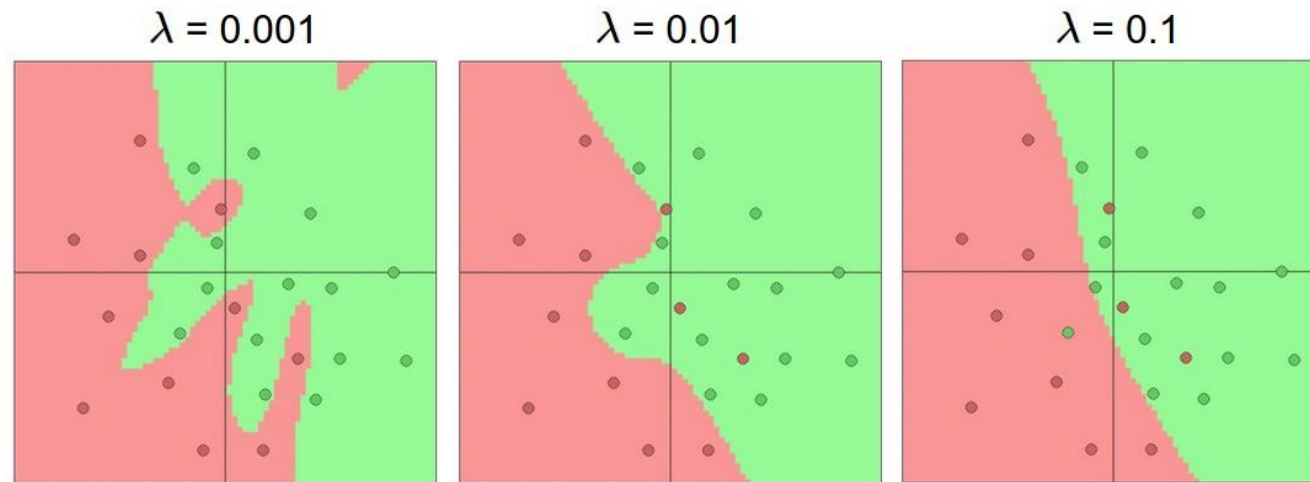
- Regularization은 overfitting 방지 효과가 있습니다
  - Logistic model 뿐 아니라 machine learning 알고리즘들에도 해당

$$\text{L1 cost} = \sum_i^n \left( y_i - \frac{1}{1 + \exp \left( -(\beta_0 + \beta_1 x_{i1} + \dots \beta_p x_{ip}) \right)} \right)^2 + \lambda \sum_j^p |\beta_j|$$

$$\text{L2 cost} = \sum_i^n \left( y_i - \frac{1}{1 + \exp \left( -(\beta_0 + \beta_1 x_{i1} + \dots \beta_p x_{ip}) \right)} \right)^2 + \lambda \sum_j^p |\beta_j|^2$$

# L2 Regularization

- L2 regularization은 경계면을 날카롭지 않게 만듭니다
  - $\beta_i$  중에서 크기가 유독 큰 값이 없도록 만들기 때문
  - Regression에 L2 penalty를 줄 경우, ridge regression이라 부름



Feed forward neural network의 예시이지만, Logistic Regression 역시 동일한 모습을 보입니다

(출처) <http://cs231n.github.io/neural-networks-1/>

# L2 Regularization

- Regularization은 overfitting 방지 효과가 있습니다
  - Logistic model 뿐 아니라 machine learning 알고리즘들에도 해당

$$\text{L2 cost} = \sum_i^n \left( y_i - \frac{1}{1 + \exp(-(\beta_0 + \beta_1 x_{i1} + \dots \beta_p x_{ip}))} \right)^2 + \boxed{\lambda} \sum_j^p |\beta_j|$$

$x$ 를 이용하여  $y$ 를 얼마나 잘 예측하는가?

결정단면이 얼마나 복잡한가?

(trade-off)

좀 더 복잡한 결정다면을 이용하면  
 $y$ 를 잘 맞출 수 있니?

# L1 Regularization

---

- L1 regularization은 입력변수 계수  $\beta_j$  중 일부를 0에 가깝게 만듭니다
  - 중요한 변수를 선택하는 효과
  - 몇 개의 변수만 선택적으로 0이 아닌 계수를 가지도록 하기 때문에 sparse modeling 이라고도 부릅니다

$$cost = \sum_i^n \left( y_i - \frac{1}{1 + \exp(-(\beta_0 + \beta_1 x_{i1} + \dots \beta_p x_{ip}))} \right)^2 + \lambda \sum_j^p |\beta_j|$$

# L1 Regularization

- Regularization은 overfitting 방지 효과가 있습니다
  - Logistic model 뿐 아니라 machine learning 알고리즘들에도 해당

$$\text{L1 cost} = \sum_i^n \left( y_i - \frac{1}{1 + \exp(-(\beta_0 + \beta_1 x_{i1} + \dots \beta_p x_{ip}))} \right)^2 + \boxed{\lambda} \sum_j^p |\beta_j|$$

$x$ 를 이용하여  $y$ 를 얼마나 잘 예측하는가?

모델이 몇 개의 변수를 이용하는가?

(trade-off)  
변수를 좀 더 이용하면  
 $y$ 를 잘 맞출 수 있니?

# L1 Regularization

---

- L1은 L0 과 L2의 특징을 모두 지니고 있습니다
  - L0 처럼 소수의 변수를 선택합니다
  - L2 처럼 미분을 통하여 coefficients의 학습이 가능합니다
    - L0는 미분이 되지 않습니다
  - $\beta_j$ 크기까지 제한하는 효과까지 있습니다
    - L0 는 크기를 제한하지는 않습니다

$$cost = \sum_i^n \left( y_i - \frac{1}{1 + \exp(-(\beta_0 + \beta_1 x_{i1} + \dots \beta_p x_{ip}))} \right)^2 + \lambda \sum_j^p |\beta_j|$$

# L1 Regularization

---

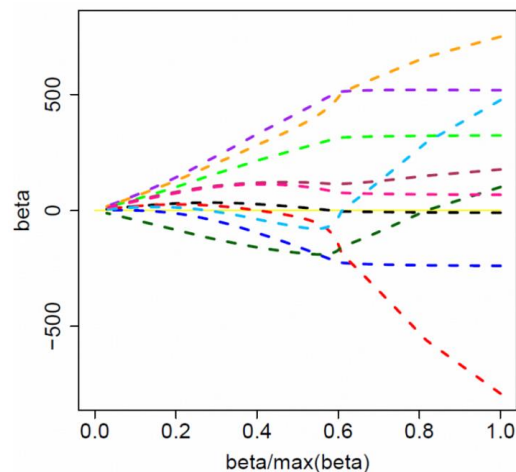
- L1 regularization을 이용한 방법을 LASSO라 부르며, Lasso regression은 중요한 변수를 데이터 기반으로 추출합니다

$$\text{cost} = \sum_i^n \left( y_i - \frac{1}{1 + \exp(-(\beta_0 + \beta_1 x_{i1} + \dots \beta_p x_{ip}))} \right)^2 + \lambda \sum_j^p |\beta_j|$$

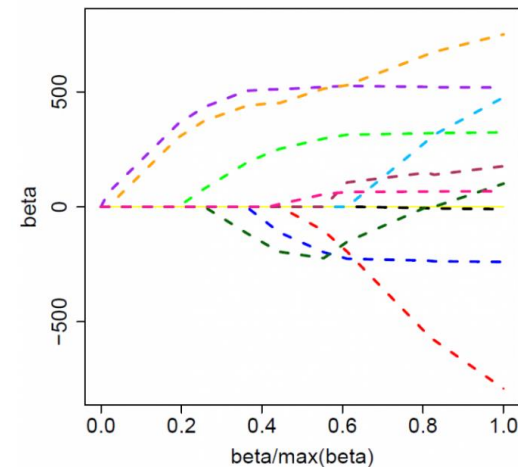
- $\lambda$ 의 크기에 따라서 모델이 선택하는 변수의 개수와 종류가 달라집니다
  - $\lambda$  값이 작을수록 (regularization을 덜 할수록) 더 많은 변수를 사용

# L1 Regularization

- LASSO path는  $\lambda$ 에 따른  $\beta$ 의 변화를 시각적으로 표현합니다
  - L2 은  $\lambda$ 가 작아짐에 따라 ( $=\beta/\max(\beta)$ 가 커짐에 따라) 여러  $\beta$ 의 값이 동시에 증가합니다
  - L1 은 step function처럼  $\lambda$ 가 어느 정도 작아져야 새로운 변수가 이용됩니다 (해당 변수의  $\beta$ 의 크기가 0이 아니게 됩니다)



LASSO path of L2



LASSO path of L1



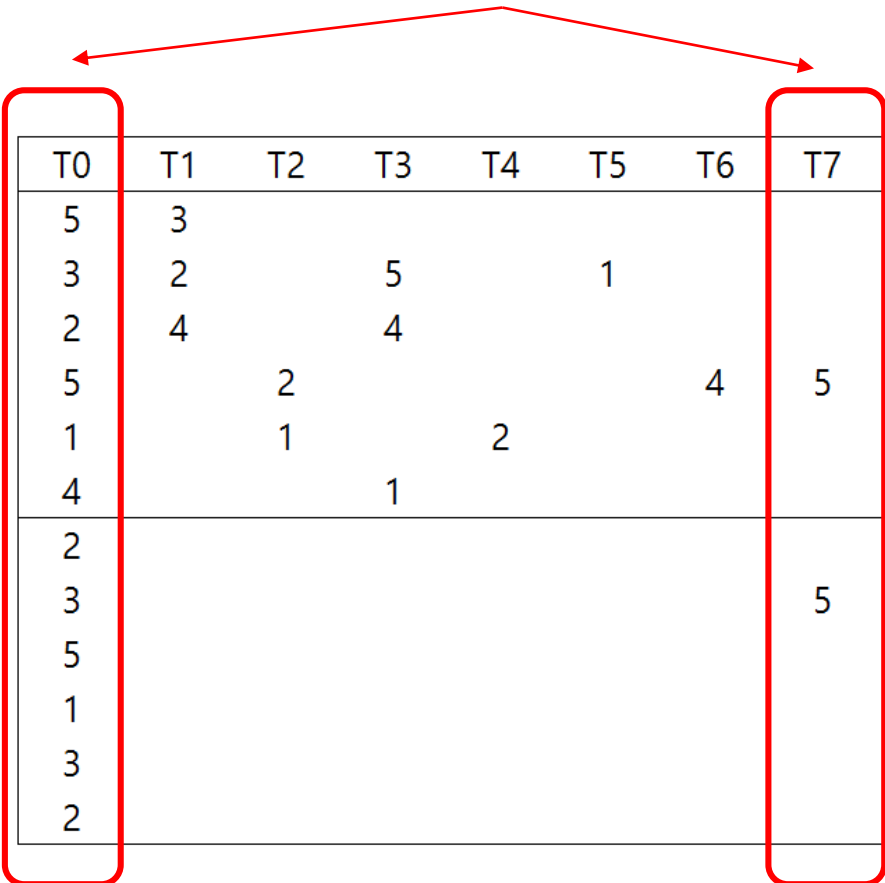
# L1 Regularization

---

- L1 regularization (LASSO model)은 성능을 저하하지 않으면서도, 모델이 이용하는 features의 개수를 최소화 하려 합니다
- Term frequency vector로 표현된 문서의 종류를 분류하는 문제에서는, 적은 수의 단어를 이용하여 문서를 잘 맞추는 모델을 선택합니다

# L1 Regularization

Classification에 전혀 도움이 되지 않는 features (terms)  
[-은, -는, -에서, ...] 와 같은 문법 관련 단어들



y	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
0	5	3													
0	3	2		5		1			2						
0	2	4		4											
0	5		2				4	5	3						
0	1		1		2										
0	4			1											
1	2								2					2	
1	3							5					4	4	
1	5								1	1		3			
1	1								2			2			3
1	3								4		1		2	1	1
1	2								4				1		2

# L1 Regularization

T1, T2만 이용하여도  $y=0$ 의 5/6을 인식할 수 있음

y	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
0	5	3													
0	3	2		5		1			2						
0	2	4		4											
0	5		2				4	5	3						
0	1		1		2										
0	4			1											
1	2								2					2	
1	3							5					4	4	
1	5								1	1		3			
1	1								2			2			3
1	3								4		1		2	1	1
1	2								4				1		2

# L1 Regularization

여유가 된다면 ( $=\lambda$  가 작다면, =classification 성능에 더 집중해도 된다면)  
T3까지 이용하면  $y=0$ 을 완벽히 인식할 수 있음

y	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
0	5	3													
0	3	2		5		1			2						
0	2	4		4											
0	5		2				4	5	3						
0	1		1		2										
0	4			1											
1	2								2					2	
1	3							5					4	4	
1	5								1	1		3			
1	1								2			2			3
1	3								4		1		2	1	1
1	2								4				1		2

# L1 Regularization

T8은  $y=1$ 일 때 더 많이 등장하지만, classification이 잘 되지 않을 가능성이 있음

y	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
0	5	3													
0	3	2		5		1			2						
0	2	4		4											
0	5		2				4	5	3						
0	1		1		2										
0	4			1											
1	2								2					2	
1	3							5					4	4	
1	5								1	1		3			
1	1								2			2			3
1	3								4		1		2	1	1
1	2								4				1		2

# L1 Regularization

T11, T13, T14를 이용하면  $y=1$ 을 완벽히 인식할 수 있고,  
이 문제는 {T1, T2, T3, T11, T13, T14}만 이용해도 잘 풀림

y	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
0	5	3													
0	3	2		5		1			2						
0	2	4		4											
0	5		2				4	5	3						
0	1		1		2										
0	4			1											
1	2								2					2	
1	3							5					4	4	
1	5								1	1		3			
1	1								2			2			3
1	3								4		1		2	1	1
1	2								4				1		2

# L1 Regularization

---

- 이러한 LASSO model의 성질을 이용하면 키워드를 추출할 수 있습니다
- LASSO 선택하는 모델은 두 가지 조건을 만족합니다
  - (1) 분별력이 좋으면서
  - (2) 많은 문서에서 등장한 단어를 우선적으로 선택
- 몇 번 등장하지 않은 단어는 분별력은 좋을 수 있지만, L1 cost를 높입니다
- 동일한 분별력을 가질 때에는 좀 더 자주 등장한 단어를 선택합니다  
(문서에 대한 coverage가 높은 단어가 선택될 가능성이 더 높습니다)

# Keyword Extraction

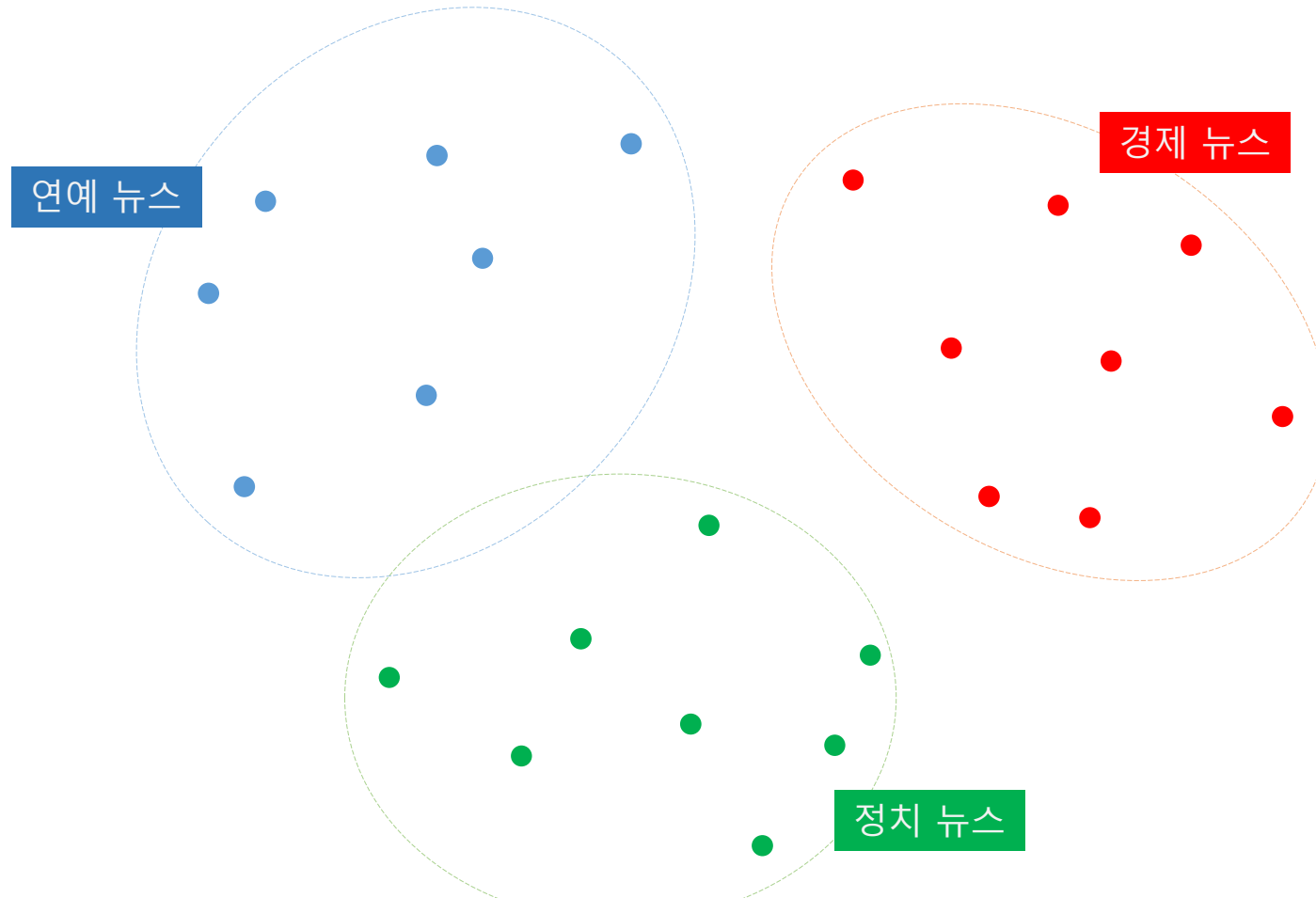
---

- 키워드는 명확히 정의된 개념이 아닙니다
  - 빈도수가 높으면? 많은 문서에서 등장했으면?
  - 우리는 키워드가 무엇인지부터 정의를 해야 합니다
- 문서 종류를 명확히 구분하면서도 그 종류의 문서에서 자주 등장한 단어를 키워드로 정의할 수도 있습니다
  - 변별력이 좋고 많은 문서에서 등장하는 단어 집합이면, 적은 수의 단어로 해당 문서 집합을 표현할 수 있습니다.
  - 이는 LASSO가 추출하는 features와 keywords의 정의가 일치합니다



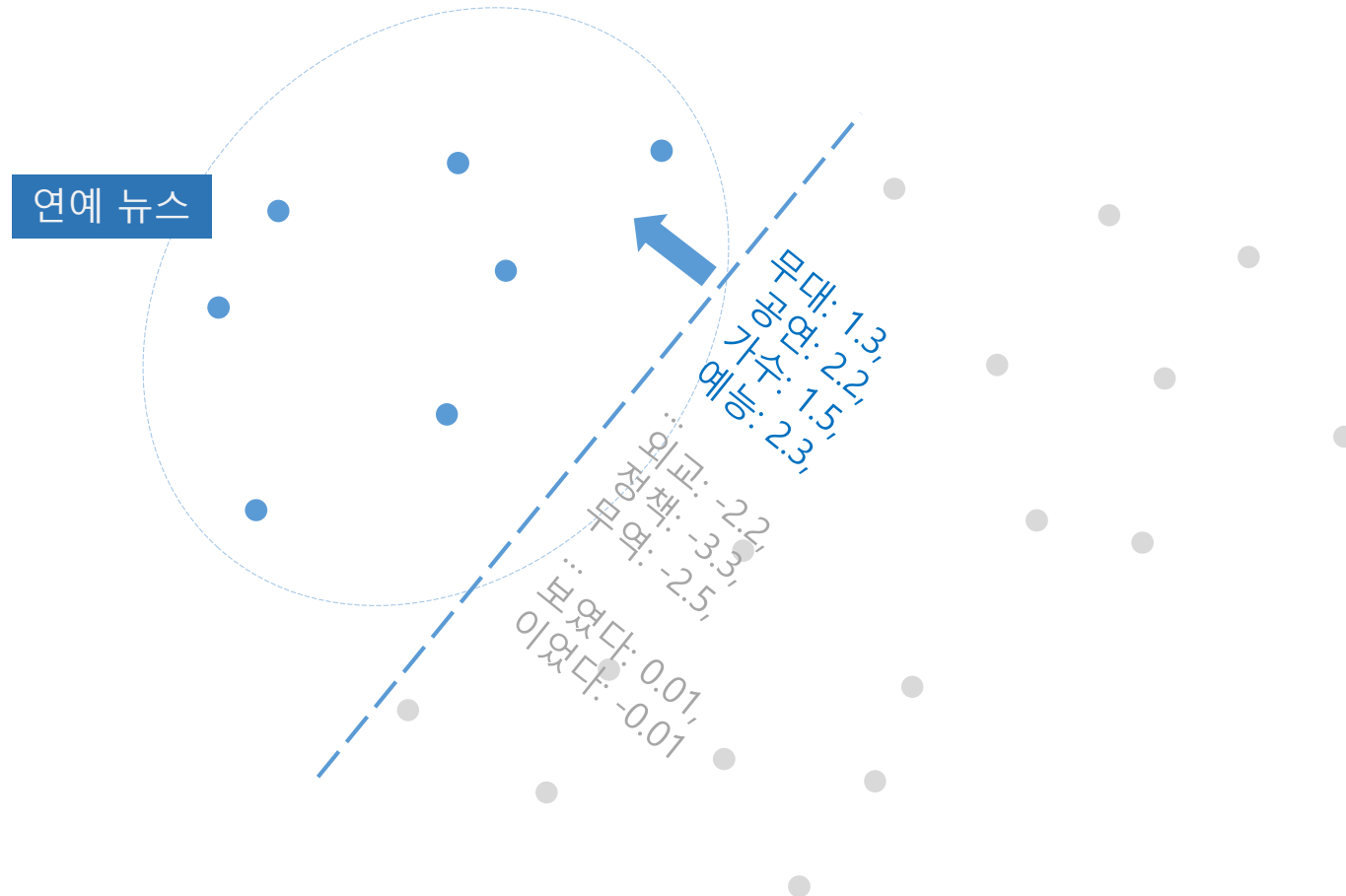
# Keyword Extraction

- 소수의 단어만으로 해당 클래스를 잘 구분할 수 있다면, 그 단어는 키워드



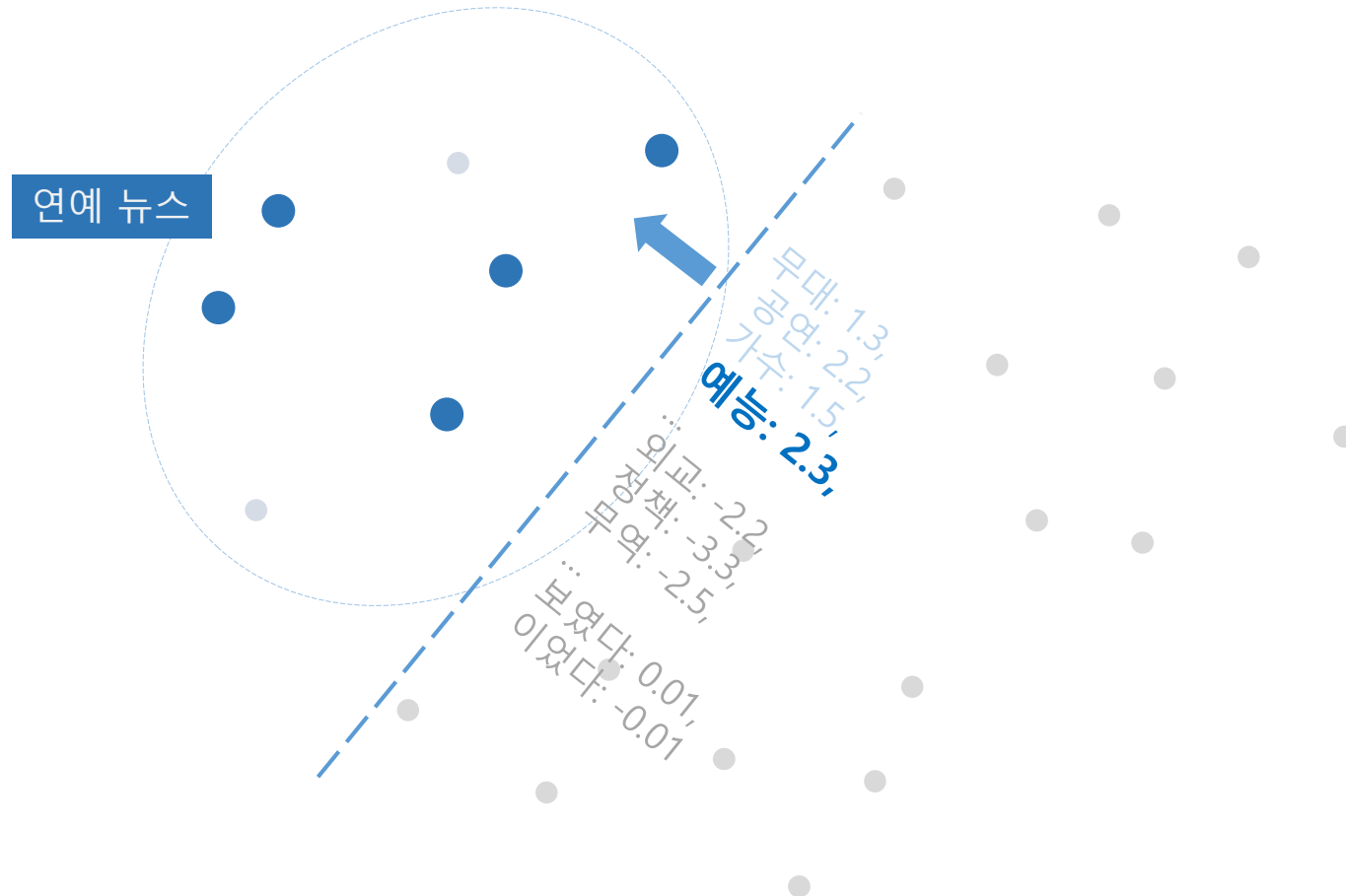
# Keyword Extraction

- 연예 뉴스를 잘 구분하는 단어는, 다른 클래스에서는 잘 등장하지 않지만, 연예 뉴스에서 집중적으로 등장하는 단어입니다



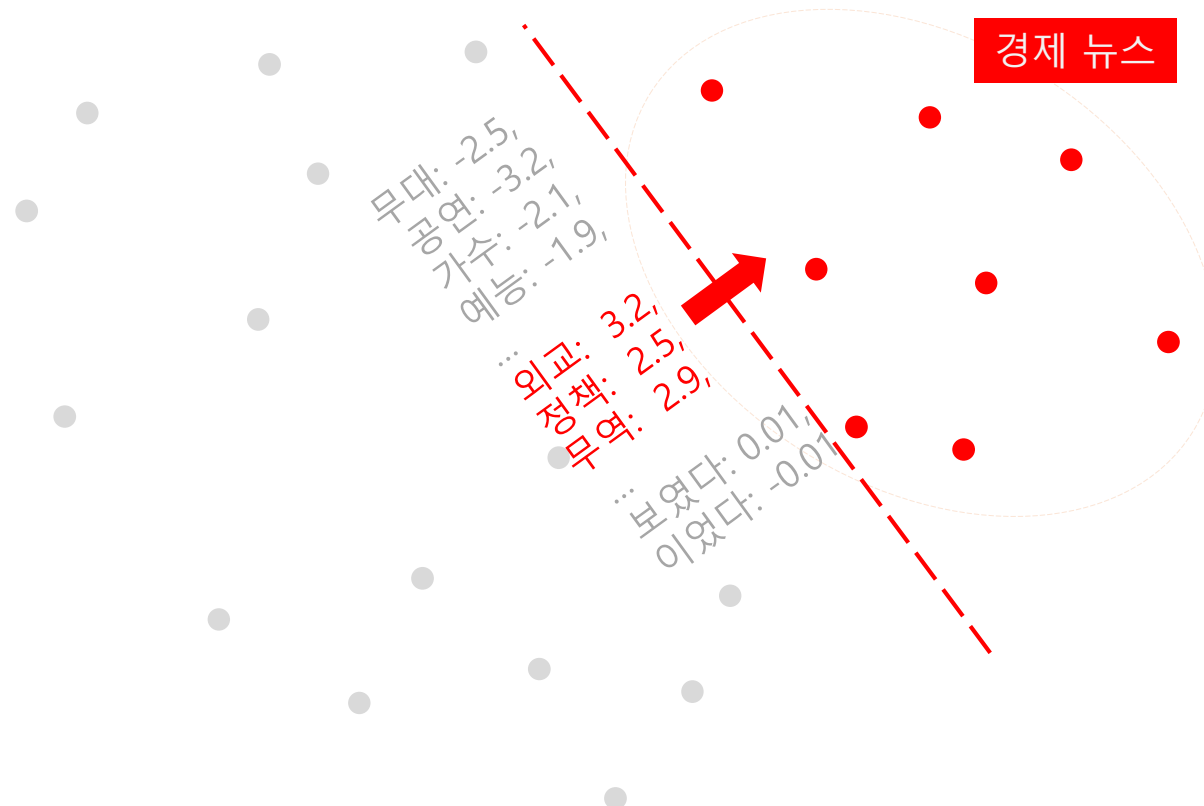
# Keyword Extraction

- 그 중에서도 "예능"처럼 많은 연예 뉴스에서 등장하는 (coverage가 높은) 단어라면, 이는 "연예 뉴스" 집단을 대표한다고 말할 수 있습니다



# Keyword Extraction

- 동일한 방식으로 “경제 뉴스” vs 다른 모든 뉴스를 구분하는 단어를 LASSO regression 을 이용하여 키워드로 추출



# LASSO Logistic Regression

---

- scikit-learn의 Logistic Regression 모델

```
from sklearn.linear import LogisticRegression
```

```
model = LogisticRegression(C=1, penalty='l1')
```

```
model.fit(train_x, train_y)
```

- penalty를 L2 로 바꿔주면 됩니다

# LASSO Logistic Regression

---

- LASSO는 L1 비용 계수에 의하여 사용하는 features 개수가 달라집니다
  - 비용 계수를 조절하며, 모델이 이용하는 단어의 개수를 조절할 수 있습니다

```
from sklearn.linear import LogisticRegression

model = LogisticRegression(C=1 → 10, penalty='l1')
model.fit(train_x, train_y)
```

# Keyword extraction for positive documents

---

- LASSO regression을 이용하면 positive set의 키워드 추출이 가능합니다

```
def lasso_keyword(word, C=20, topk=50):  
    if not (word in _word2int):  
        return []  
  
    x_train, y_train = get_train_data(word)  
    logistic_l1 = LogisticRegression(penalty='l1', C=C)  
    logistic_l1.fit(x_train, y_train)  
  
    sorted_coefficients = sorted(enumerate(logistic_l1.coef_.reshape(-1)), key=lambda x: -x[1])  
    keywords = [word_idx for word_idx, coef in sorted_coefficients[:topk] if coef > 0.001]  
    keywords = [int2word(word_idx) for word_idx in keywords]  
    return keywords
```

# Keyword extraction for positive documents

- LASSO regression을 이용하면 positive set의 키워드 추출이 가능합니다

```
def lasso_keyword(word, C=20, topk=50):  
    if not (word in _word2int):  
        return []  
  
    x_train, y_train = get_train_data(word)  
    logistic_l1 = LogisticRegression(penalty='l1', C=C)  
    logistic_l1.fit(x_train, y_train)  
  
    sorted_coefficients = sorted(enumerate(logistic_l1.coef_.reshape(-1)), key=lambda x: -x[1])  
    keywords = [word_idx for word_idx, coef in sorted_coefficients[:topk] if coef > 0.001]  
    keywords = [int2word(word_idx) for word_idx in keywords]  
    return keywords
```

\* 자세한 코드는 jupyter tutorials