

NLTK 6: 텍스트 분류 학습

박수지

컴퓨터언어학연구실

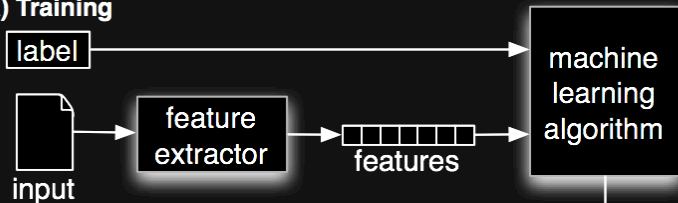
2017년 10월 17일

Learning to Classify Text

- 1 Supervised Classification
 - Gender Identification
 - Choosing The Right Features
 - Document Classification
- 2 Further Examples of Supervised Classification
- 3 Evaluation
 - The Test Set
 - Accuracy
 - Precision and Recall
 - Confusion Matrices
 - Cross-Validation
- 4 Maximum Entropy modeling assignment

Supervised Classification

(a) Training



(b) Prediction



Supervised Classification

지도 분류 (Supervised Classification)

- 1 훈련 (Training): **정답이 표시된**(=이미 분류된) 자료 → 패턴 학습
 - ▶ 자질 (Feature): 자료의 패턴을 식별하고 기술하는 기준
 - ▶ 알고리즘 (Algorithm): 자질값으로부터 분류 결과를 계산해 내는 방법
- 2 실험 (Test)=예측 (Prediction): 학습된 패턴 → 새로운 자료 분류

주의 분류 가능한 범주의 목록은 미리 정해져 있다.
→ 분류 중에 새로운 목록을 만들 수 없다!

Supervised Classification

Gender Identification

이름 성별 분류: 자질 추출

- 패턴
- a,e,i로 끝남 → 여성
 - k,o,r,s,t로 끝남 → 남성

자질 이름의 끝 글자가 무엇인가?

함수 정의 주어진 이름의 자질값을 뽑기 (변수명 `gender_features`)

입력 문자열 (변수명 `word`)

출력 딕셔너리

키 `'last_letter'`

값 `word[-1]` (`word`의 마지막 한 글자)

```
1 >>> def gender_features(word):
2 ...     return {'last_letter': word[-1]}
3 >>> gender_features('Shrek')
4 {'last_letter': 'k'}
```

Supervised Classification

Gender Identification

이름 성별 분류: 자료 확보

출전 `nltk.corpus.names`

- 코퍼스 설치: `nltk.download()`

유형 리스트 (변수명 `labeled_names`)

원소 2-tuple: (이름, 레이블=성별)

예시 `(u'Aaron', 'male')`
`(u'Zoe', 'female')`

순서 섞기 `random.shuffle()`

```
1 >>> from nltk.corpus import names
2 >>> labeled_names = (
3 ... [(name, 'male') for name in names.words('male.txt')] +
4 ... [(name, 'female') for name in names.words('female.txt')])
5 >>> import random
6 >>> random.shuffle(labeled_names)
```

Supervised Classification

Gender Identification

이름 성별 분류: 분류기 구성 → 훈련 집합에서 학습

1 자질 집합 구성: 리스트 (변수명 `featuresets`)

원소 2-tuple: (자질=딕셔너리, 레이블=성별)

예시 (`{'last_letter': n}, 'male'`)
(`{'last_letter': e}, 'female'`)

▶ 주의: Aaron, Zoe 등 구체적인 이름이 자질로 환원됨

2 코퍼스 분할

평가 집합 자질 집합의 앞 500개 원소

훈련 집합 자질 집합의 나머지 원소 (7,944 - 500 = 7,444개)

3 분류기 (변수명 `classifier`)

알고리즘 Naive Bayes: `nltk.NaiveBayesClassifier`

```
1 >>> featuresets = [(gender_features(n), gender)
2 ...                 for (n, gender) in labeled_names]
3 >>> train_set, test_set = featuresets[500:], featuresets[:500]
4 >>> classifier = nltk.NaiveBayesClassifier.train(train_set)
```

Supervised Classification

Gender Identification

이름 성별 분류: 분류기 적용

함수 `분류기.classify()`

입력 자질 딕셔너리

출력 레이블=성별

```
1 | >>> classifier.classify(gender_features('Neo'))  
2 | 'male'  
3 | >>> classifier.classify(gender_features('Trinity'))  
4 | 'female'
```

이름 성별 분류: 분류기 성능 평가

함수 `nltk.classify.accuracy()`

입력 분류기, 실험 집합

출력 정확도 (실험 집합에서 분류기 적용 결과가 정답과 일치하는 것의 개수/실험 집합 원소 개수)

결과 0.77 → 우연히 맞출 0.5보다 높다!

```
1 | >>> print(nltk.classify.accuracy(classifier, test_set))  
2 | 0.77
```


Supervised Classification

Gender Identification

이름 성별 분류: 자질 검토

자질 정보량 평가

함수 `분류기.show_most_informative_features()`

입력 개수 `n`

출력 정보량이 높은 상위 `n`개 자질

예시 `a`로 끝나면 여성일 확률이 남성일 확률보다 33.2배 높다.

```
1 >>> classifier.show_most_informative_features(5)
2 Most Informative Features
3 last_letter = 'a' female : male    = 33.2 : 1.0
4 last_letter = 'k'   male : female = 32.6 : 1.0
5 last_letter = 'p'   male : female = 19.7 : 1.0
6 last_letter = 'v'   male : female = 18.6 : 1.0
7 last_letter = 'f'   male : female = 17.3 : 1.0
```

Supervised Classification

Choosing The Right Features

이름 성별 분류: 자질 추가

목표 기존 모형 성능(정확도 0.77) 개선

자질 끝 글자("last_letter") 외에 추가할 정보

- 첫 글자("first_letter")
- 글자 개수("count(letter)")
- 글자 포함 여부("has(letter)")

```
1 >>> def gender_features2(name):
2 ...     features = {}
3 ...     features["first_letter"] = name[0].lower()
4 ...     features["last_letter"] = name[-1].lower()
5 ...     for letter in 'abcdefghijklmnopqrstuvwxyz':
6 ...         features["count({})".format(letter)] = name.lower().count(letter)
7 ...         features["has({})".format(letter)] = (letter in name.lower())
8 ...     return features
9 >>>
10 >>> gender_features2('John')
11 {'count(j)': 1, 'has(d)': False, 'count(b)': 0, ...}
```

Supervised Classification

Choosing The Right Features

이름 성별 분류: 성능 비교

`gender_features()` v. `gender_features2()`

- 1 `gender_features2()`를 바탕으로 자질 집합 구성
- 2 동일한 학습 집합에서 새 분류기 학습
- 3 새 분류기를 동일한 실험 집합에 적용
- 4 정확도 평가: $0.768 < 0.77$

→ 자질이 추가되었지만 결과가 나쁘다!

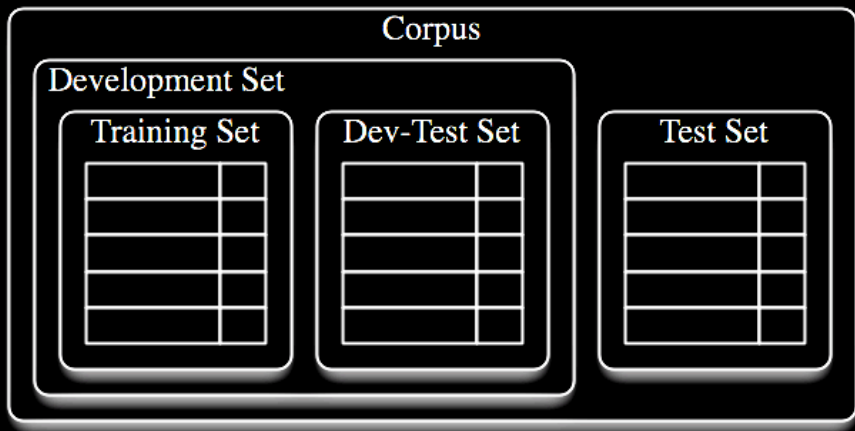
```
1 >>> featuresets = [(gender_features2(n), gender)
2 ... for (n, gender) in labeled_names]
3 >>> train_set, test_set = featuresets[500:], featuresets[:500]
4 >>> classifier = nltk.NaiveBayesClassifier.train(train_set)
5 >>> print(nltk.classify.accuracy(classifier, test_set))
6 0.768
```

Supervised Classification

Choosing The Right Features

이름 성별 분류: 오류 분석

개발 집합 (development set) 도입



Supervised Classification

Choosing The Right Features

이름 성별 분류: 오류 분석

코퍼스 재분할

실험 1-500번째 이름

개발-평가 501-1500번째 이름

훈련 나머지 이름

→ 분류기 성능을 개발-평가 집합에서 먼저 평가: 정확도 0.75

```
1 >>> train_names = labeled_names[1500:]
2 >>> devtest_names = labeled_names[500:1500]
3 >>> test_names = labeled_names[:500]
4 >>> train_set = [(gender_features(n), gender)
5 ... for (n, gender) in train_names]
6 >>> devtest_set = [(gender_features(n), gender)
7 ... for (n, gender) in devtest_names]
8 >>> test_set = [(gender_features(n), gender)
9 ... for (n, gender) in test_names]
10 >>> classifier = nltk.NaiveBayesClassifier.train(train_set)
11 >>> print(nltk.classify.accuracy(classifier, devtest_set))
12 0.75
```

Supervised Classification

Choosing The Right Features

이름 성별 분류: 오류 분석

개발-평가 집합에서 정답 (변수명 tag) 과 예측 (변수명 guess) 이 다른 목록

```
1 | >>> errors = []
2 | >>> for (name, tag) in devtest_names:
3 | ...     guess = classifier.classify(gender_features(name))
4 | ...     if guess != tag:
5 | ...         errors.append( (tag, guess, name) )
```

Supervised Classification

Choosing The Right Features

이름 성별 분류: 오류 분석

출력된 결과의 패턴 관찰 → 마지막 두 글자가 중요하다!

yn 여성

ch 남성

```
1 >>> for (tag, guess, name) in sorted(errors):
2     ...     print('correct={:<8} guess={:<8s} name={:<30}'.
3     ...     format(tag, guess, name))
4 correct=female   guess=male     name=Abigail
5     ...
6 correct=female   guess=male     name=Cindelyn
7     ...
8 correct=female   guess=male     name=Katheryn
9 correct=female   guess=male     name=Kathryn
10    ...
11 correct=male    guess=female    name=Aldrich
12    ...
13 correct=male    guess=female    name=Mitch
14    ...
15 correct=male    guess=female    name=Rich
16
```

Supervised Classification

Choosing The Right Features

이름 성별 분류: 자질 추출기 수정

자질 목록 수정

- 마지막 한 글자
- 마지막 두 글자

결과 정확도 0.782

→ 마지막 글자만 사용했을 때 (0.77) 보다 높아졌다!

```
1 >>> def gender_features(word):
2 ...     return {'suffix1': word[-1:],
3 ...             'suffix2': word[-2:]}
4 >>> train_set = [(gender_features(n), gender)
5 ...               for (n, gender) in train_names]
6 >>> devtest_set = [(gender_features(n), gender)
7 ...                 for (n, gender) in devtest_names]
8 >>> classifier = nltk.NaiveBayesClassifier.train(train_set)
9 >>> print(nltk.classify.accuracy(classifier, devtest_set))
10 0.782
```


Supervised Classification

Document Classification

영화평 분류: 자료 확보

출전 `nltk.corpus.movie_reviews`

유형 리스트 (변수명 `documents`)

원소 2-tuple: (문서 내 단어 목록, 레이블=긍정.부정)

예시 `([u'plot', ..., u'echoes'], u'neg')`
`([u'if', u'there', ...], u'pos')`

순서 섞기 `random.shuffle()`

```
1 >>> from nltk.corpus import movie_reviews
2 >>> documents = [(list(movie_reviews.words(fileid)), category)
3 ...               for category in movie_reviews.categories()
4 ...               for fileid in movie_reviews.fileids(category)]
5 >>> random.shuffle(documents)
```

Supervised Classification

Document Classification

영화평 분류: 자질 추출

자질 고빈도 단어 (총 2,000개)

- 빈도 분포 추출 함수 `nltk.FreqDist()`

유형 딕셔너리

키 `contains(word)`

값 단어가 문서에 있는지 여부 (True/False)

```
1 >>> all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
2 >>> word_features = list(all_words)[:2000] [1]
3 >>>
4 >>> def document_features(document): [2]
5 ...     document_words = set(document) [3]
6 ...     features = {}
7 ...     for word in word_features:
8 ...         features['contains({})'.format(word)] = (word in document_words)
9 ...     return features
10 >>> print(document_features(movie_reviews.words('pos/cv957_8737.txt')))
11 {'contains(waste)': False, 'contains(lot)': False, ...}
```

Supervised Classification

Document Classification

이름 성별 분류: 분류기 구성 → 학습 → 평가

- 1 자질 집합 구성
- 2 코퍼스 분할 — 평가: 훈련=100:1900
- 3 분류기 알고리즘: Naive Bayes
- 4 성능: 정확도 0.81

▶ 주요 자질: outstanding, seagal, wonderfully, damon, wasted, ...

```
1 >>> featuresets = [(document_features(d), c) for (d,c) in documents]
2 >>> train_set, test_set = featuresets[100:], featuresets[:100]
3 >>> classifier = nltk.NaiveBayesClassifier.train(train_set)
4 >>> print(nltk.classify.accuracy(classifier, test_set))
5 0.81
6 >>> classifier.show_most_informative_features(5)
7 Most Informative Features
8     contains(outstanding) = True    pos : neg = 11.1 : 1.0
9         contains(seagal) = True    neg : pos = 7.7 : 1.0
10    contains(wonderfully) = True    pos : neg = 6.8 : 1.0
11        contains(damon) = True    pos : neg = 5.9 : 1.0
12    contains(wasted) = True    neg : pos = 5.8 : 1.0
```

Learning to Classify Text

- 1 Supervised Classification
 - Gender Identification
 - Choosing The Right Features
 - Document Classification
- 2 Further Examples of Supervised Classification
- 3 Evaluation
 - The Test Set
 - Accuracy
 - Precision and Recall
 - Confusion Matrices
 - Cross-Validation
- 4 Maximum Entropy modeling assignment

Learning to Classify Text

- 1 Supervised Classification
 - Gender Identification
 - Choosing The Right Features
 - Document Classification
- 2 Further Examples of Supervised Classification
- 3 Evaluation
 - The Test Set
 - Accuracy
 - Precision and Recall
 - Confusion Matrices
 - Cross-Validation
- 4 Maximum Entropy modeling assignment

Evaluation

The Test Set

실험 집합 구성 방법: (1) 순서 뒤섞기

출전 브라운 코퍼스 뉴스 카테고리

분할 문장 순서 뒤섞기 `random.shuffle(tagged_sents)`

평가 집합 문장 목록 앞 10%

훈련 집합 문장 목록 뒤 90%

- 문제점
- 1 평가 집합과 훈련 집합의 장르 동일
→ 다른 장르에 적용되는지 확인 불가능
 - 2 평가 집합과 훈련 집합의 문서 중복
→ 훈련 효과와 정답 단순 암기의 구별 불가능

```
1 >>> import random
2 >>> from nltk.corpus import brown
3 >>> tagged_sents = list(brown.tagged_sents(categories='news'))
4 >>> random.shuffle(tagged_sents)
5 >>> size = int(len(tagged_sents) * 0.1)
6 >>> train_set, test_set = tagged_sents[size:], tagged_sents[:size]
```

Evaluation

The Test Set

실험 집합 구성 방법: (2) 문서 파일 아이디로 구별

출전 브라운 코퍼스 뉴스 카테고리

분할 문서 파일 아이디로 구별 (`file_ids`)

평가 집합 파일 아이디 목록 중 앞 10%

훈련 집합 파일 아이디 목록 중 뒤 90%

개선점 ① 평가 집합과 훈련 집합의 문서 중복 문제 해결
→ 분류기의 신뢰성 확보

문제점 ① 평가 집합과 훈련 집합의 장르 동일
→ 다른 장르에 적용되는지 확인 불가능

```
1 >>> file_ids = brown.fileids(categories='news')
2 >>> size = int(len(file_ids) * 0.1)
3 >>> train_set = brown.tagged_sents(file_ids[size:])
4 >>> test_set = brown.tagged_sents(file_ids[:size])
```

Evaluation

The Test Set

실험 집합 구성 방법: (3) 문서 장르로 구별

출전 브라운 코퍼스 뉴스 카테고리

분할 카테고리 값으로 구별

평가 집합 픽션 장르 문장 목록

훈련 집합 뉴스 장르 문장 목록

- 개선점
- 1 평가 집합과 훈련 집합의 문서 중복 문제 해결
→ 분류기의 신뢰성 확보
 - 2 평가 집합과 훈련 집합의 장르 동일 문제 해결
→ 분류기의 일반성 확인 가능

```
1 | >>> train_set = brown.tagged_sents(categories='news')
2 | >>> test_set = brown.tagged_sents(categories='fiction')
```


Evaluation

Accuracy

다양한 분류 성능 평가 척도

- 정확도 (Accuracy)
- 정밀도 (Precision)
- 재현도 (Recall)
- F-점수 (F-score)
- ...

정확도 (Accuracy)

전체 자료에서 제대로 예측한 것이 얼마나 되는가?

Evaluation

Accuracy

예시: 스팸 메일 감지

	스팸 취급	덜짱 취급
진짜 스팸		
진짜 덜짱		

$$\text{정확도} = \frac{\text{진짜 스팸} \text{ 옳게 분류된 개수} + \text{진짜 덜짱} \text{ 옳게 분류된 개수}}{\text{옳게 분류된 개수 총합}} = \frac{\text{실제=예측인 것들의 개수}}{\text{전체 개수}}$$

Evaluation

Accuracy

예시: 스팸 메일 감지

분류 방법

- 양성 (Positive): 스팸
- 음성 (Negative): 스팸 아님 → 정상

분류 결과

- 스팸을 스팸이라고 함: 😡 (진양성 True positive)
- 정상을 스팸이라고 함: 😡 (위양성 False positive — Type I Error)
- 스팸을 정상이라고 함: 😡 (위음성 False negative — Type II Error)
- 정상을 정상이라고 함: 😊 (진음성 True negative)

Evaluation

Accuracy

정확도 측정의 문제

한쪽 부류의 빈도가 압도적으로 높은 경우

→ 무조건 한쪽 부류로 예측해도 정확도가 높다.

Evaluation

Accuracy

예시: 스팸 메일 감지

이메일 1,000통 중 990통이 스팸이고 10통이 멀쩡할 때

→ 무조건 스팸으로 취급해도 99%의 정확도를 얻는다!

- 스팸을 스팸이라고 함: 😡 990
- 정상을 스팸이라고 함: 😡 10
- 스팸을 정상이라고 함: 😡 0 (무조건 스팸이라고 하기로 했으므로)
- 정상을 정상이라고 함: 😊 0 (무조건 스팸이라고 하기로 했으므로)

$$\text{정확도} = \frac{\text{😡} + \text{😊}}{\text{😡} + \text{😡} + \text{😡} + \text{😊}} = \frac{990 + 0}{990 + 10 + 0 + 0} = \frac{990}{1000} = 99\%$$

→ 받아야 하는 메일 10통을 모두 놓치고도 99%의 정확도로 칭찬받는다!

Evaluation

Precision and Recall

정밀도와 재현도

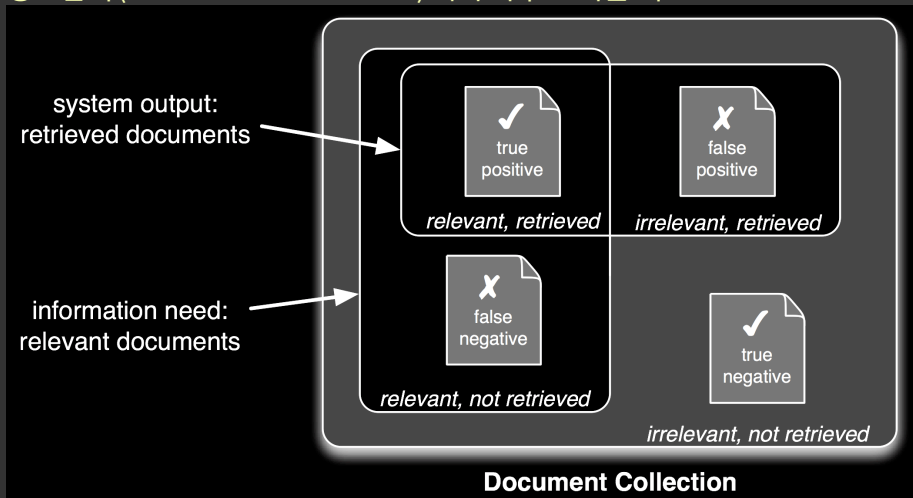
- 정밀도 (Precision): **찾은 것** 중에서 제대로 찾은 것이 얼마나 되는가?
- 재현도 (Recall): **찾아야 하는 것** 중에서 제대로 찾은 것이 얼마나 되는가?
- F-Score: 정밀도와 재현도의 조화평균

찾지 말아야 하는 것을 찾지 않은 경우 (진음성, True Negative) 무시

Evaluation

Precision and Recall

정보검색 (Information Retrieval) 에서 자주 쓰이는 척도



Evaluation

Precision and Recall

예시: 스팸 메일 감지

	스팸 취급	멀쩡 취급
진짜 스팸	 990	 0
진짜 멀쩡	 10	 0

스팸 메일 기준 척도

“제대로 찾은 것” 스팸을 스팸이라고 함  = 990

“찾은 것” 스팸이라고 함  +  = 990 + 10 = 1000

“찾아야 하는 것” 진짜 스팸임  +  = 990 + 0 = 990

정밀도 $990/1000 = 99\%$

재현도 $990/990 = 100\%$






Evaluation

Precision and Recall

예시: 스팸 메일 감지

	멀쩡 취급	스팸 취급
진짜 멀쩡	 0	 10
진짜 스팸	 0	 990

정상 메일 기준 척도

“제대로 찾은 것”	정상을 정상이라고 함	 = 0
“찾은 것”	정상이라고 함	 +  = 0 + 0 = 0
“찾아야 하는 것”	진짜 정상임	 +  = 10 + 0 = 10

정밀도 $0/0 =$ (분모가 0이므로 계산 불가, 관례상 0%으로도 표현)

재현도 $0/10 = 0\%$

Evaluation

Confusion Matrices

세 가지 이상의 범주로 분류하는 경우

```
1 >>> def tag_list(tagged_sents):
2 ...     return [tag for sent in tagged_sents for (word, tag) in sent]
3 >>> def apply_tagger(tagger, corpus):
4 ...     return [tagger.tag(nltk.tag.untag(sent)) for sent in corpus]
5 >>> gold = tag_list(brown.tagged_sents(categories='editorial'))
6 >>> test = tag_list(apply_tagger(t2,
7 ...                             brown.tagged_sents(categories='editorial')))
8 >>> cm = nltk.ConfusionMatrix(gold, test)
9 >>> print(cm.pretty_format(sort_by_count=True, show_percents=True,
10 ...                        truncate=9))
```

		N	I	A	J	N	V	N
		N	N	T	J	S	B	P
NN	<11.8%>	0.0%	.	0.2%	.	0.0%	0.3%	0.0%
IN	0.0%	<9.0%>	.	.	.	0.0%	.	.
AT	.	.	<8.6%>
JJ	1.7%	.	.	<3.9%>	.	.	0.0%	0.0%
.	<4.8%>	.	.	.
NNS	1.5%	<3.2%>	.	0.0%
,	<4.4%>	.
VB	0.9%	.	.	0.0%	.	.	<2.4%>	.
NP	1.0%	.	.	0.0%	.	.	.	<1.8%>

Evaluation

Cross-Validation

교차검증법 (Cross-validation)

코퍼스 N 등분 → 훈련집합-실험집합 분할 N 회 반복

예시: 10-fold cross-validation

- 1 코퍼스 10등분: C_1, C_2, \dots, C_{10}
- 2 분류기 학습-적용-평가 10회 반복
 - ▶ 훈련집합 C_2, C_3, \dots, C_{10} / 실험집합 C_1
 - ▶ 훈련집합 C_1, C_3, \dots, C_{10} / 실험집합 C_2
 - ▶ ...
 - ▶ 훈련집합 C_1, \dots, C_8, C_{10} / 실험집합 C_9
 - ▶ 훈련집합 C_1, C_2, \dots, C_9 / 실험집합 C_{10}
- 3 총 10회 성능 척도의 평균 계산

Learning to Classify Text

- 1 Supervised Classification
 - Gender Identification
 - Choosing The Right Features
 - Document Classification
- 2 Further Examples of Supervised Classification
- 3 Evaluation
 - The Test Set
 - Accuracy
 - Precision and Recall
 - Confusion Matrices
 - Cross-Validation
- 4 Maximum Entropy modeling assignment

Maximum Entropy modeling assignment

Word Sense Disambiguation

http://www-rohan.sdsu.edu/~gawron/compling/course_core/assignments/new_maxent_assignment.htm

과제

단어 의미 중의성 해소 (WSD: Word Sense Disambiguation)

분류 대상 동음이의어나 다의어의 의미

예시 영어 형용사 'hard'

- ① difficult to do
 - ▶ I find it hard to believe that ...
- ② potent (as in "the hard stuff")
 - ▶ They look old and dumpy, with stolid, hard faces.
- ③ physically resistant to denting, bending, or scratching
 - ▶ Water becomes stiff and hard as clear stone.

Word Sense Disambiguation

코퍼스

파일 senseval-hard.xml

규모 4,333문장

전처리 소문자화, 품사 부착

레이블 단어-품사 (word="hard-a"), 의미 ("HARD1"), 위치 ("20")

```
예시 <senseval_instance word="hard-a" sense="
      HARD1" position="20">
```

‘_‘ he_PRP may_MD lose_VB all_DT
popular_JJ support_NN ,_, but_CC
someone_NN has_VBZ to_TO kill_VB
him_PRP to_TO defeat_VB him_PRP and_CC
that_DT 's_VBZ hard_JJ to_TO do_VB ._.
,, ,,

</senseval_instance>

Maximum Entropy modeling assignment

Word Sense Disambiguation

자질 추출기

파일 `call_extract_event.py`

자질 문장 내 단어 존재 여부 (1: 있음, 0: 없음)

입력 코퍼스 파일 `senseval-hard.xml`

출력 이벤트 파일 `senseval-hard.evt` (1문장 = 1이벤트)

예시 BEGIN EVENT

HARD1

soft_JJ 0

has_VBZ 1

...

only_RB 0

END EVENT

문제 자질을 어떻게 선택할 것인가?

Maximum Entropy modeling assignment

Word Sense Disambiguation

분류기 학습

파일 `call_maxent.py`

모형 최대엔트로피 모형 `nlk.classify.maxent.MaxentClassifier`
• (IIS 알고리즘: improved iterative scaling)

반복 횟수 50회

코퍼스 분할 4,333개 중 4,100개 사용 → 순서 뒤섞기

훈련 집합 앞 90% 문장

실험 집합 뒤 10% 문장

결과 실험 집합 410개 문장 분류

주의 NLTK 3.0에서 변경된 함수명 변경

`classify.batch_classify()`

→ `classify.classify_many()`

Maximum Entropy modeling assignment

Word Sense Disambiguation

분류기 학습 (계속)

예시 | Testing classifier...

2 | Accuracy: 0.8220

3 | Total: 410

4 |

5 |

6 | Label

Precision

Recall

7 |

8 | HARD1

0.822

1.000

9 |

9 | HARD2

0.000

0.000

10 |

10 | HARD3

0.000

0.000

11 |

12 |

13 | Label

Num Corr

14 |

14 | HARD1

337

15 |

15 | HARD2

0

16 |

16 | HARD3

0

Maximum Entropy modeling assignment

Word Sense Disambiguation

분류기 학습 (계속)

해석 (전체 정확도) = (HARD1 정밀도) = $0.8220 = 337/410$
→ 410개 문장 전체가 HARD1으로 분류됨

```
1 Testing classifier...
2 Accuracy: 0.8220
3 Total: 410
4
5
6 Label                Precision      Recall
7 -----
8 HARD1                0.822        1.000
9 HARD2                0.000        0.000
10 HARD3               0.000        0.000
11
12
13 Label                Num Corr
14 HARD1                337
15 HARD2                 0
16 HARD3                 0
```

Maximum Entropy modeling assignment

Word Sense Disambiguation

```
1 def extract_vocab(event_list, n=100):
2     # Google's stoplist with most preps removed. "and" added, word added
3     stopwords = [ 'I',      'a',      'an',      'are',      'as',      'and',
4                   'be',      'com',      'how',      'is',      'it',      'of', 'or',
5                   'that',    'the',      'this',    'was',      'what',
6                   'when',    'where', 'who',      'will',    'with',
7                   'the',     'www',    'was']
8     vocab = Counter()
9     for (s_inst, sense) in event_list:
10         for (i,item) in enumerate(s_inst.context):
11             if i == int(s_inst.position):
12                 continue
13             (item, wd, pos) = get_lex_components(item)
14             if wd in stopwords:
15                 continue
16             if pos in ['PRP', 'IN', 'CC', 'DT']:
17                 continue
18             vocab[item] += 1
19     il = vocab.items()
20     il.sort(key=lambda x: x[1], reverse=True)
21     il = il[:n]
22     vocab = dict(il)
23     return vocab
```

Supervised Classification

- ① 자질 집합: [(자질 딕셔너리, 분류 결과) 튜플을 원소로 가지는 리스트]

```
1 | featsets = [({'feat1': 'val11', 'feat2': 'val21', ...}, 'class1'),
2 |              ({'feat1': 'val12', 'feat2': 'val22', ...}, 'class2'),
3 |              ...,
4 |              ({'feat1': 'val1N', 'feat2': 'val2N', ...}, 'classN')]
```

- ② 집합 분할: 일반적으로 학습:실험=9:1로 분할

```
1 | N = len(featsets)
2 | train_set = featsets[:int(N*0.9)]
3 | test_set = featsets[int(N*0.9):]
```

- ③ 분류기 학습: 분류기 클래스의 train 메소드로 학습 집합에서 학습

```
1 | classifier = nltk.classify.naivebayes.NaiveBayesClassifier
2 | classifier = nltk.classify.maxent.MaxentClassifier
3 | classifier.train(train_set)
4 | dir(nltk.classify)
```

- ④ 분류기 적용 및 평가: 학습한 분류기로 실험 집합을 예측한 결과 평가

```
1 | nltk.classify.accuracy(classifier, test_set))
```