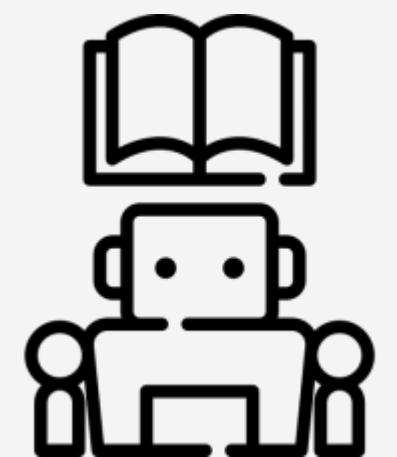


---

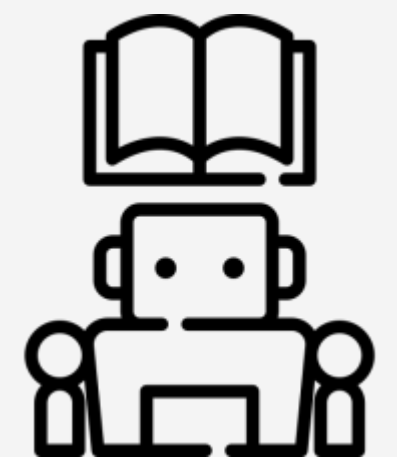
# Machine Learning Term Project #2

Spam 문자 분류하기



---

# Overview



# [Overview] - 프로젝트 방향성 이해하기

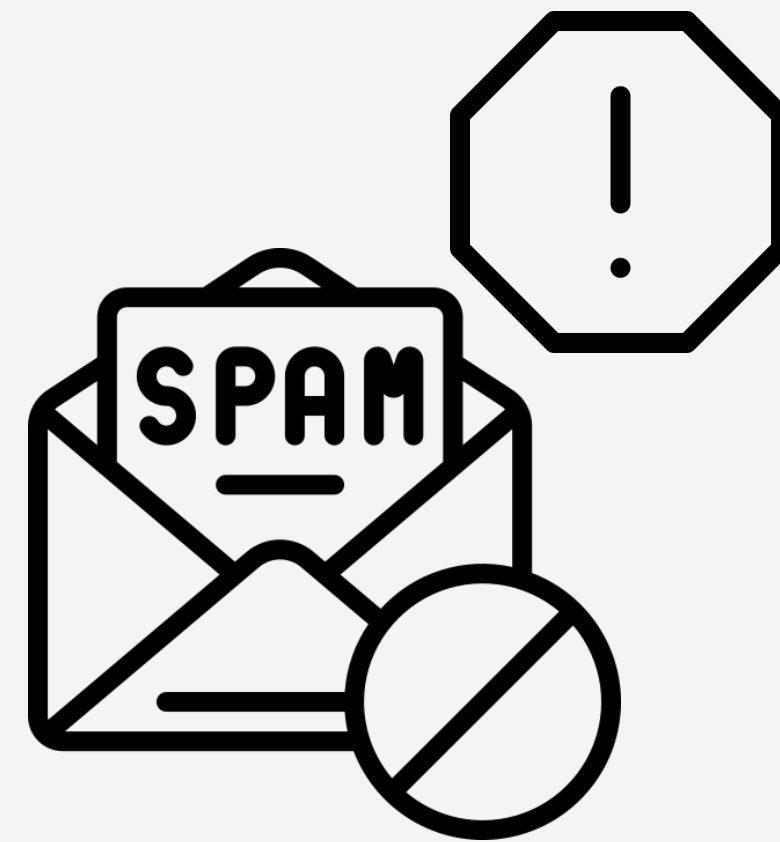
---

## 프로젝트의 목적

본 프로젝트를 통해 1D 텍스트 데이터를  
Handcrafted Feature로 기술하는 법을 알 수 있다.

Handcrafted feature란?

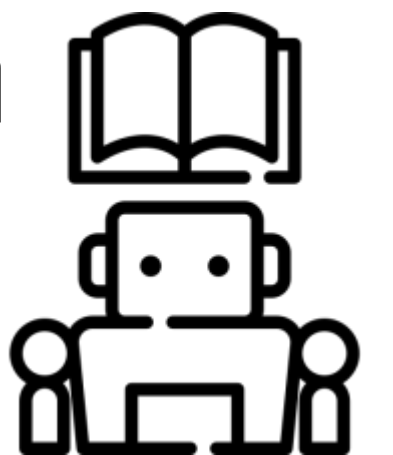
: 전문가에 의해 고안된 아이디어를 바탕으로 직접 설계된 특징



---

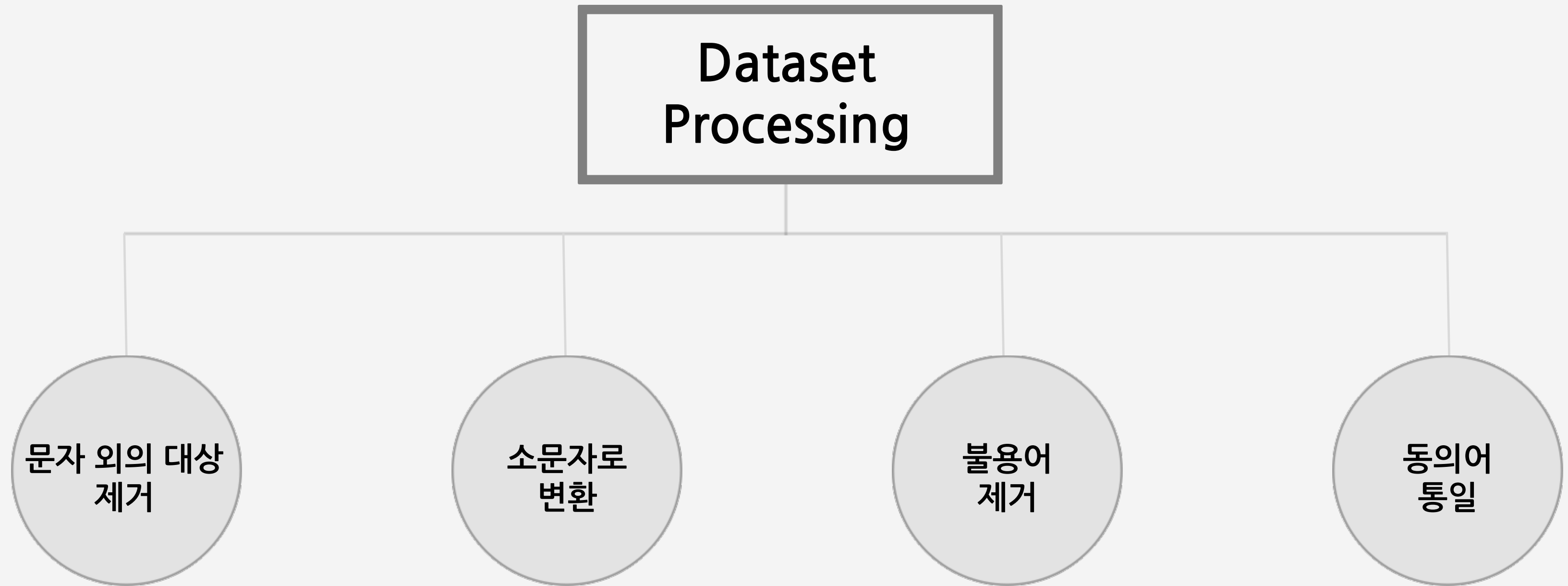
# Empty Module #1

**Reconstruction based anomaly detection**



# [Overview] - Reconstruction based anomaly detection

---



# [Empty Module #1] - Reconstruction based anomaly detection

```
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

def data_processing(text):

    # [1] re.sub 사용해 text 속 '['^A-Za-z']' 외의 문자만을 찾아내 제거한후, pre_words 변수에 저장
    pre_words = re.sub('[^A-Za-z]', ' ', text)

    # [2] pre_words의 lower 내장 함수를 이용해 대문자들은 소문자로 변경
    pre_words = pre_words.lower()

    # [3] word_tokenize 함수를 이용해 pre_word 를 토큰화하여 word를 리스트화한 후 tokenized_words변수에 저장
    tokenized_words = word_tokenize(pre_words)

    # [4] nltk 라이브러리로 다운 받은 stopwords의 "words" 내장 함수를 이용해 english 불용어를 찾아서 stops 변수에 저장
    stops = stopwords.words()

    tokenized_words_remove=[]
    for w in tokenized_words:
        # [5] [3] 에서 찾은 문자열 중 단어가 [4] 에서 찾은 불용어 속에 없을 경우, tokenized_words_remove 리스트에 append
        if w not in stops:
            tokenized_words_remove.append(w)

    stemmer = PorterStemmer()
    for i in range(len(tokenized_words_remove)):
        # [6] tokenized_words_remove의 단어를 PorterStemmer 속 stem 내장 함수를 이용해, 동일 의미를 갖는 단어를 동일한 단어로 변경하는 과정을 거친 후 다시 저장
        tokenized_words_remove[i] = stemmer.stem(tokenized_words_remove[i])

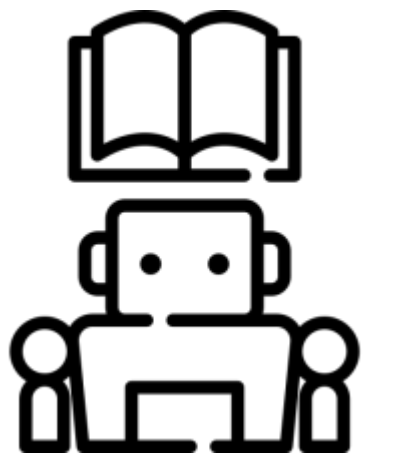
    return ( " ".join( tokenized_words_remove ) )
```



---

# Empty Module #2

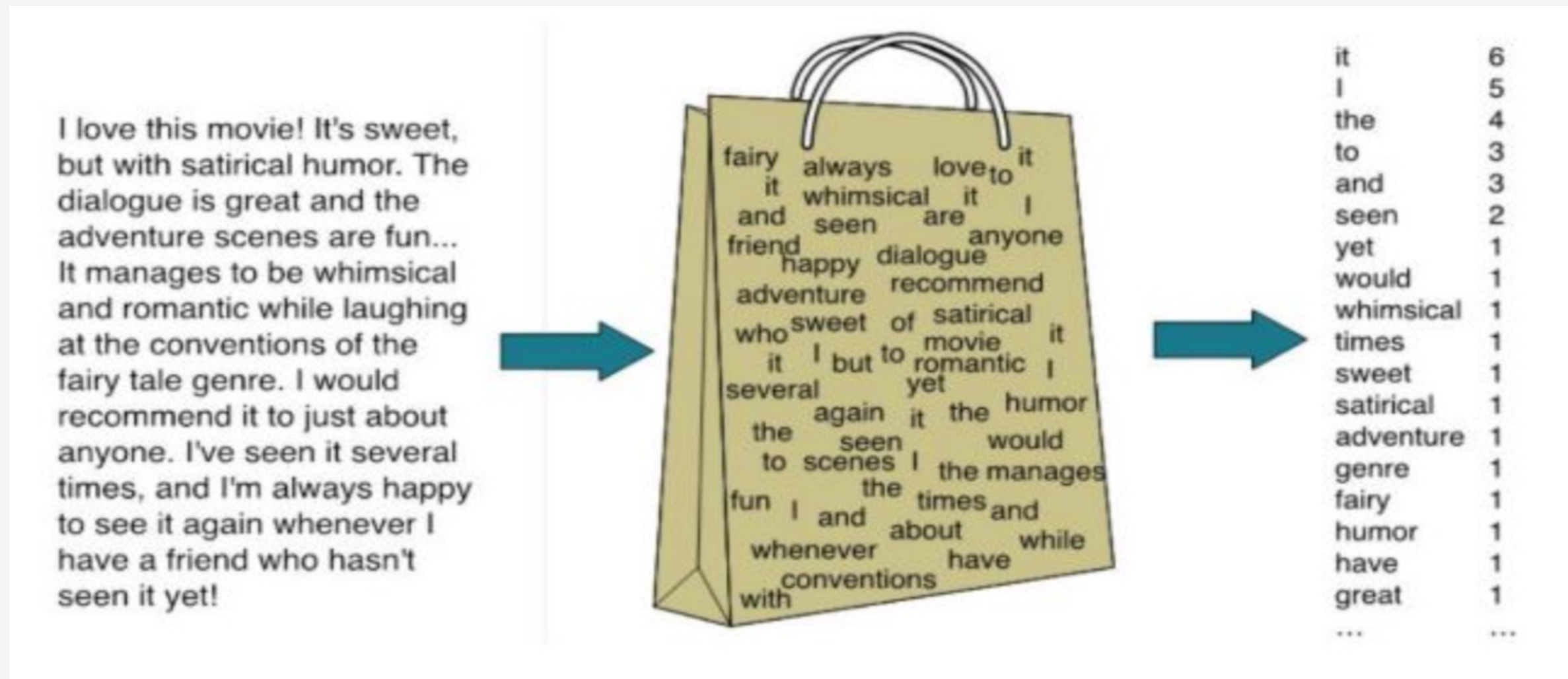
**Bag of Words**



# [Empty Module #1] - Reconstruction based anomaly detection

## Bag of Words

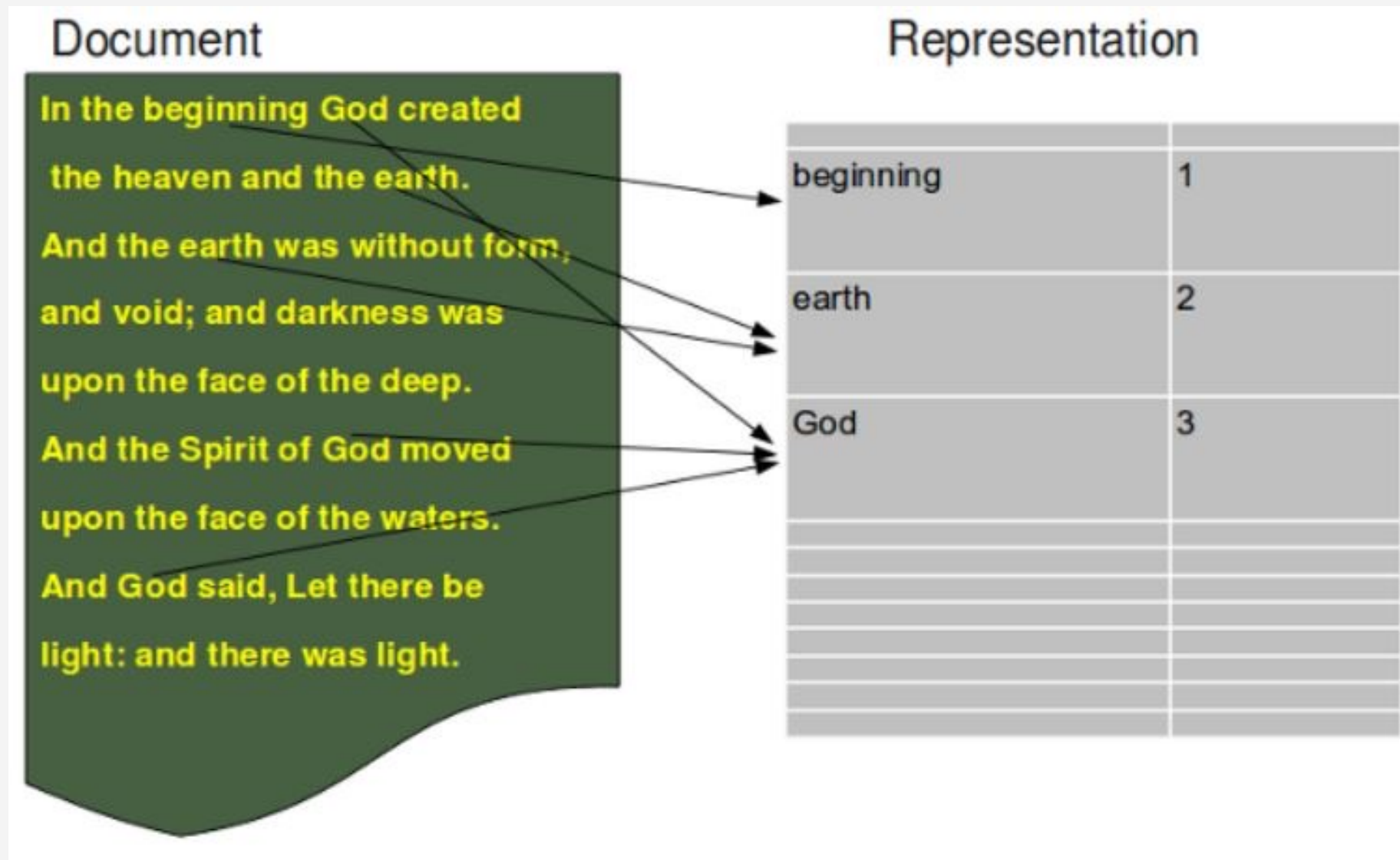
: 단어들의 출현 빈도에 집중하는 텍스트 데이터의 수치화 표현 방법





# [Empty Module #2] - Bag of words

## Bag of Words



모든 텍스트의 모든 단어 가져오기



각 단어의 발생 횟수 계산



Cluster Word 선택



**feature**  
→ Cluster Word의 발생 횟수

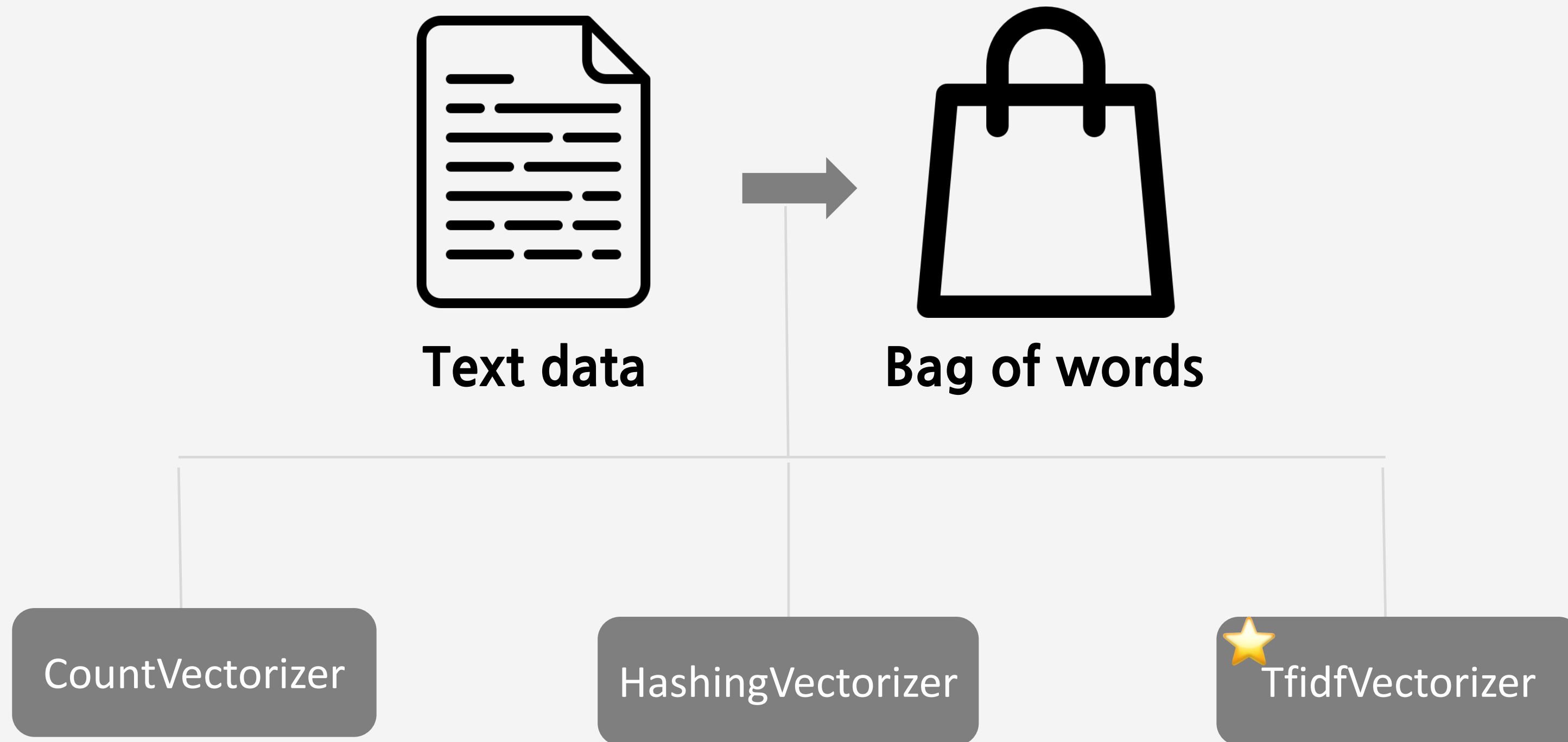


분류 진행



## [Empty Module #2] Bag of words

---



# [Empty Module #2] Bag of words

---



CountVectorizer

→ 문서 집합에서 단어 토큰을 생성하고  
각 단어의 수를 세어 BoW 인코딩 벡터를 만든다.

HashingVectorizer

→ 해시 함수(hash function)를 사용하여  
적은 메모리와 빠른 속도로 BoW 인코딩 벡터를 만든다.

TfidfVectorizer

→ 'CountVectorizer'와 비슷하지만  
TF-IDF 방식으로 단어의 가중치를 조정한 BoW 인코딩 벡터를 만든다.



# [Empty Module #2] Bag of words

---

## CountVectorizer

*# CountVectorizer*

*# [1] CountVectorizer를 정의*

```
from sklearn.feature_extraction.text import CountVectorizer  
vectorizer = CountVectorizer()
```

*# [2] X\_train 과 X\_test를 numpy array로 변환 후 데이터 타입을 "U"로 변경해 저장*

```
X_train = np.array(X_train).astype("U")
```

```
X_test = np.array(X_test).astype("U")
```

*# [3] CountVectorizer를 이용해 X\_train은 학습 및 변환(fit\_transform)을 하고, X\_test는 변환(transform)을 진행*

```
X_train_features = vectorizer.fit_transform(X_train)
```

```
X_test_features = vectorizer.transform(X_test)
```



# [Empty Module #2] Bag of words

---

## HashingVectorizer

```
# HashingVectorizer
```

```
# [1] HashingVectorizer 정의
```

```
from sklearn.feature_extraction.text import HashingVectorizer  
vectorizer = HashingVectorizer()
```

```
# [2] X_train 과 X_test를 numpy array로 변환 후 데이터 타입을 "U"로 변경해 저장
```

```
X_train = np.array(X_train).astype("U")
```

```
X_test = np.array(X_test).astype("U")
```

```
# [3] HashingVectorizer 이용해 X_train은 학습 및 변환(fit_transform)을 하고, X_test는 변환(transform)을 진행
```

```
X_train_features = vectorizer.fit_transform(X_train)
```

```
X_test_features = vectorizer.transform(X_test)
```



# [Empty Module #2] Bag of words

---



## TfidfVectorizer

```
# TfidfVectorizer
```

```
# [1] TfidfVectorizer 정의
```

```
from sklearn.feature_extraction.text import TfidfVectorizer  
vectorizer = TfidfVectorizer()
```

```
# [2] X_train 과 X_test를 numpy array로 변환 후 데이터 타입을 "U"로 변경해 저장
```

```
X_train = np.array(X_train).astype("U")  
X_test = np.array(X_test).astype("U")
```

```
# [3] TfidfVectorizer 이용해 X_train은 학습 및 변환(fit_transform)을 하고, X_test는 변환(transform)을 진행
```

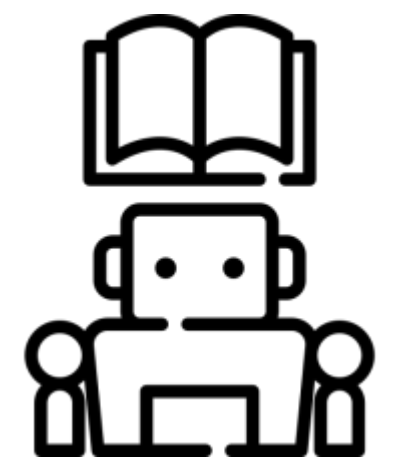
```
X_train_features = vectorizer.fit_transform(X_train)  
X_test_features = vectorizer.transform(X_test)
```



---

# Empty Module #3

**SVM: classifier**



# [Empty Module #3] - SVM: classifier

---

## Support Vector Classifier

→ SVM을 통해 margin을 최대화하기 위한 결정 경계 찾기

```
from sklearn.svm import SVC

# [1] SVC 선언 (베이스라인 에서 gamma="auto" 사용 )
svc=SVC(gamma='auto')

# [2] X_train_features과 y_train으로 SVC 학습진행 후, X_test_features로 predict 진행
svc.fit(X_train_features, y_train)

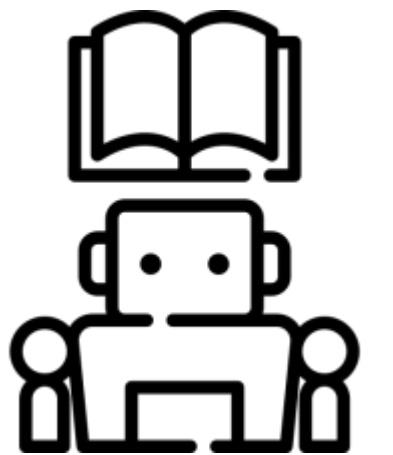
# [3] y_pred에 predict한 결과값 저장
y_pred= svc.predict(X_test_features)
```





---

# Hyperparameter Tuning



# Hyperparameter Tuning

## Vectorizer

Hyperparameter	score
CountVectorizer(max_features = 100) SVC(gamma = 'auto')	0.94439(Baseline)
HashingVectorizer() SVC(gamma = 'auto')	0.86547
TfidfVectorizer() SVC(gamma = 'auto')	0.86547

Hyperparameter	score
CountVectorizer(max_features = 100) SVC(C = 10000)	0.96143
HashingVectorizer() SVC(C = 10000)	0.97847
✓ TfidfVectorizer() SVC(C = 10000)	0.98026



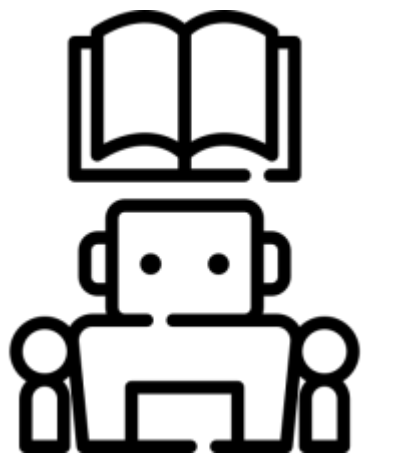
# Hyperparameter Tuning

Hyperparameter	score
CountVectorizer(max_features = 100) SVC(gamma = 'auto')	0.94439(Baseline)
HashingVectorizer() SVC(gamma = 'auto')	0.86547
TfidfVectorizer() SVC(gamma = 'auto')	0.86547
CountVectorizer(max_features = 100) SVC(C = 10000)	0.96143
HashingVectorizer() SVC(C = 10000)	0.97847
TfidfVectorizer() SVC(C = 10000)	0.98026
TfidfVectorizer(norm='l1') SVC(C = 10000)	0.98475
TfidfVectorizer(norm='l1', sublinear_tf=True) SVC(C = 10000)	0.98654
★ TfidfVectorizer(norm='l1', sublinear_tf=True, use_idf=False) SVC(C = 10000)	0.98744



---

# Final Score




# Final Score

## [2022-ML][P2]20011844\_안수경

Python · Spam 문자 분류기

Notebook Data Logs Comments (0) Settings


	Competition Notebook Spam 문자 분류기	Run 60.9s	Public Score 0.94439	Best Score <u>0.98744 V31</u>
---	-------------------------------------	--------------	-------------------------	----------------------------------

← Baseline score (0.94439)

## [2022-ML][P2]20011844\_안수경

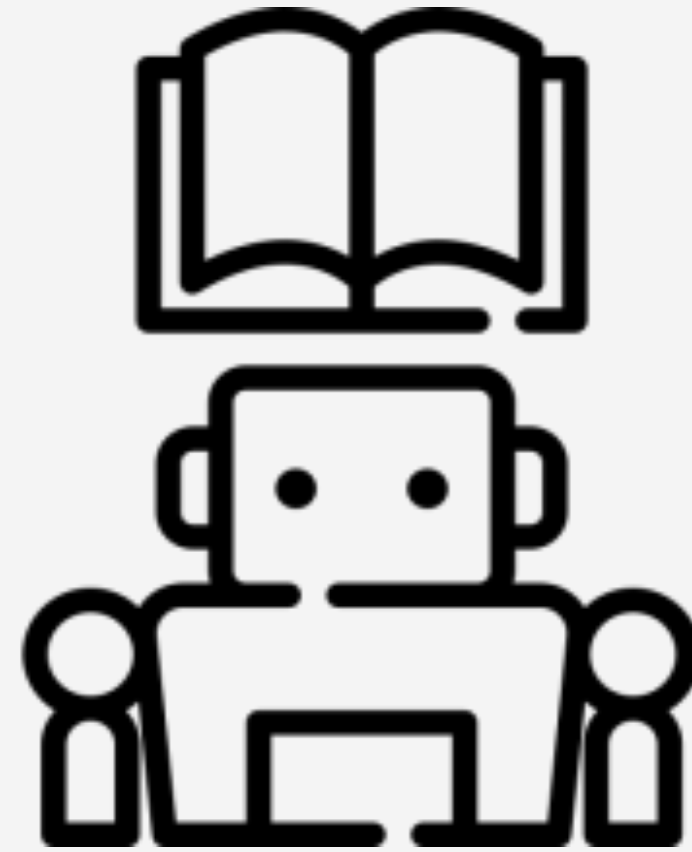
Python · Spam 문자 분류기

Notebook Data Logs Comments (0) Settings

	Competition Notebook Spam 문자 분류기	Run 69.8s	Public Score 0.98744	Best Score <u>0.98744 V31</u>
---	-------------------------------------	--------------	-------------------------	----------------------------------

← Best score (0.98744)





# THANK YOU

*Machine learning term project #2*

---

