

# 넷플릭스 경진대회

기념비적인 추천 엔진 경진 대회

## 넷플릭스 프라이즈 (Netflix Prize) 개요

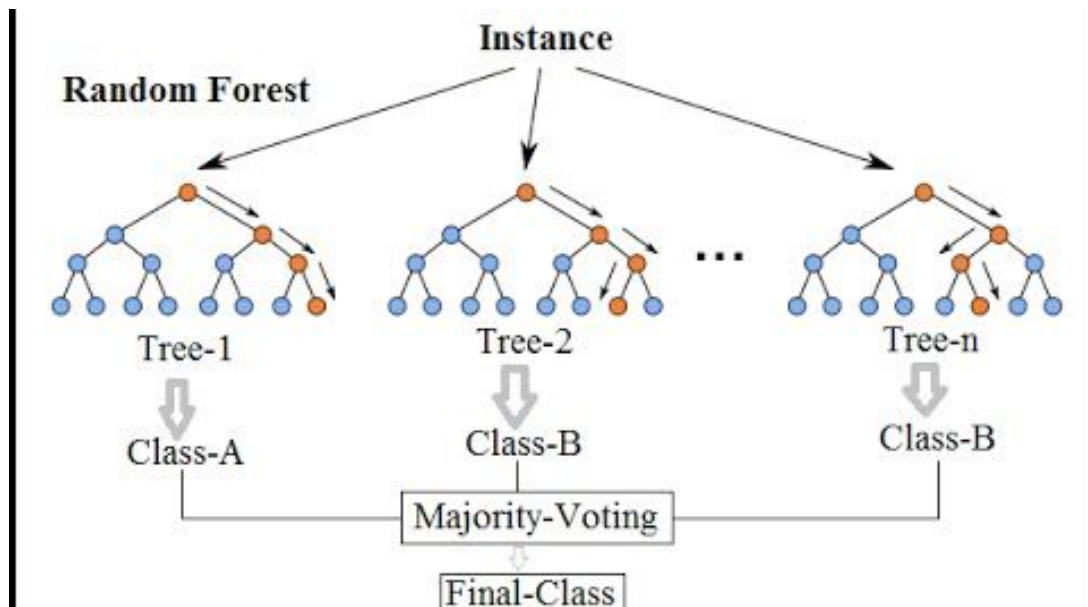
- 2006년부터 3년간 운영된 넷플릭스의 기념비적인 추천 엔진 경진대회
- 넷플릭스 추천 시스템 품질을 10% 개선하는 팀에게 \$1M 수여 약속
  - RMSE(Root Mean Square Error)가 평가 기준으로 사용됨
  - 사용자와 영화 페어에 대해 평점을 예측하고 그걸 정답값과 비교
- 넷플릭스 브랜드 인지도도 올라감
- 프라이버시 이슈도 제기됨
- 이를 기폭제로 캐글(Kaggle)과 같은 머신러닝 경진대회 플랫폼이 등장

## 넷플릭스 프라이즈 (Netflix Prize) 우승팀과 알고리즘

- 최종 우승팀은 “BellKor’s Pragmatic Chaos”
- 이 대회를 통해 협업 필터링이 한단계 발전함
  - SVD를 활용한 SVD++는 이후 굉장히 많은 분야에서 활용됨
    - SVD와 SVD++에 대해서는 뒤에서 더 자세히 살펴봄
  - 앙상블 방식의 모델들이 가장 좋은 성능을 보임
    - 너무 긴 실행시간 때문에 실제 프로덕션에서는 사용 불가
  - 다양한 알고리즘들이 논문으로 학회에서 발표됨 (SVD++ 포함)

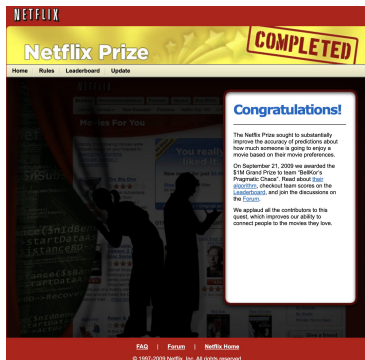
## 앙상블 (Ensemble)과 랜덤 포레스트 (Random Forest)

- 다수의 분류기를 사용해서 예측하는 방식
- 성능은 좋지만 훈련과 예측 시간이 오래 걸린다는 단점 존재



## 추천 엔진 발전 역사 (1)

- 2001년 아마존이 아이템 기반 협업 필터링 논문 발표
  - Amazon.com recommendations: item-to-item collaborative filtering
- 2006-2009년 넷플릭스 프라이즈
  - SVD를 이용한 사용자의 아이템 평점 예측 알고리즘 탄생
  - 앙상블 알고리즘이 보편화됨
  - 딥러닝의 일종이라 할 수 있는 RBM(Restricted Boltzman Machine)이 단일 모델로는 최고의 성능을 보여줌. 딥러닝의 추천에서의 사용가능성을 보임



<https://www.netflixprize.com/>

## 추천 엔진 발전 역사 (2)

- 2010년 딥러닝이 콘텐츠 기반 음악 추천에 사용되기 시작
- 2016년 딥러닝을 기반으로한 추천이 활기를 띠기 시작
  - 오토인코더 기반으로 복잡한 행렬 계산을 단순화하는 방식이 하나
  - 아이템 관련 사용자 정보를 시간순으로 인코딩하는 **RNN**을 사용하는 방식이 다른 하나
  - 아마존에서 DSSNTE라는 알고리즘을 오픈소스화한 후 **SageMaker**라는 제품으로 통합

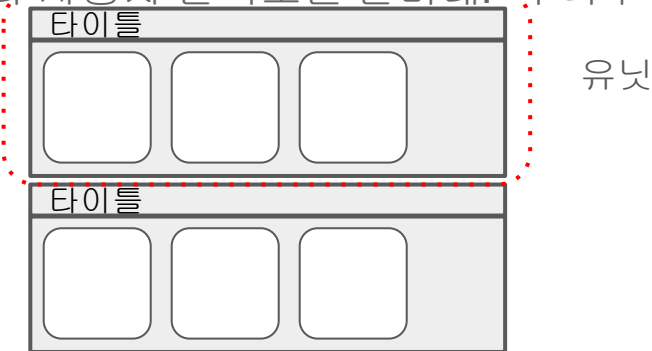
# 유데미 추천 엔진 살펴보기

유데미 추천 엔진은 어떻게 동작하는가?

## 유데미 추천 살펴보기

- 문제 정의: 학생들에게 관심있을 만한 강의를 먼저 보여주는 것!
- 추천 UI: 격자 기반의 UI
  - 다양한 추천 유닛들이 존재
    - 몇 개의 유닛을 어느 순서로 보여줄지 결정: 유닛 선택과 랭킹! 이 자체도 ML 문제
  - 페이지 생성 시간과 사용자 만족도는 반비례. 즉 너무 많은 유닛은 역효과를 가져옴

유닛의 랭킹이  
중요해짐





## 유데미 추천 살펴보기

- 온라인 강의 메타 데이터
  - 분류 체계, 태그
  - 강사에게 태그와 분류 체계 선택하게 하기, 클릭 키워드 분석, ...
- 다양한 행동 기반 추천
  - 클릭, 구매, 소비 등등

## 기본 아이디어

- 하이브리드 방식 추천:
  - 협업 필터링, 사용자 행동 기반 추천, 머신러닝 모델기반 추천
- 사용자별로 등록 확률을 기준으로 2천개의 탑 강의 목록 생성
  - 배치로 시작했다가 실시간 계산으로 변경
- 홈페이지에서의 추천은 조금더 복잡 (개인화되어 있음)
  - 유닛 후보 생성
  - 유닛 후보 랭킹
  - 위의 문제를 ML로 해결하기도 함
- 특정 강의 세부 페이지에서 추천은 아이템 중심
  - Student also bought (아이템 기반 협업 필터링)
  - Frequently bought together (별도의 co-occurrence 행렬 계산)
  - 각 유닛에서의 강의 랭킹은 개인별 등록 확률로 결정

# 인기도 기반 추천 엔진 개발

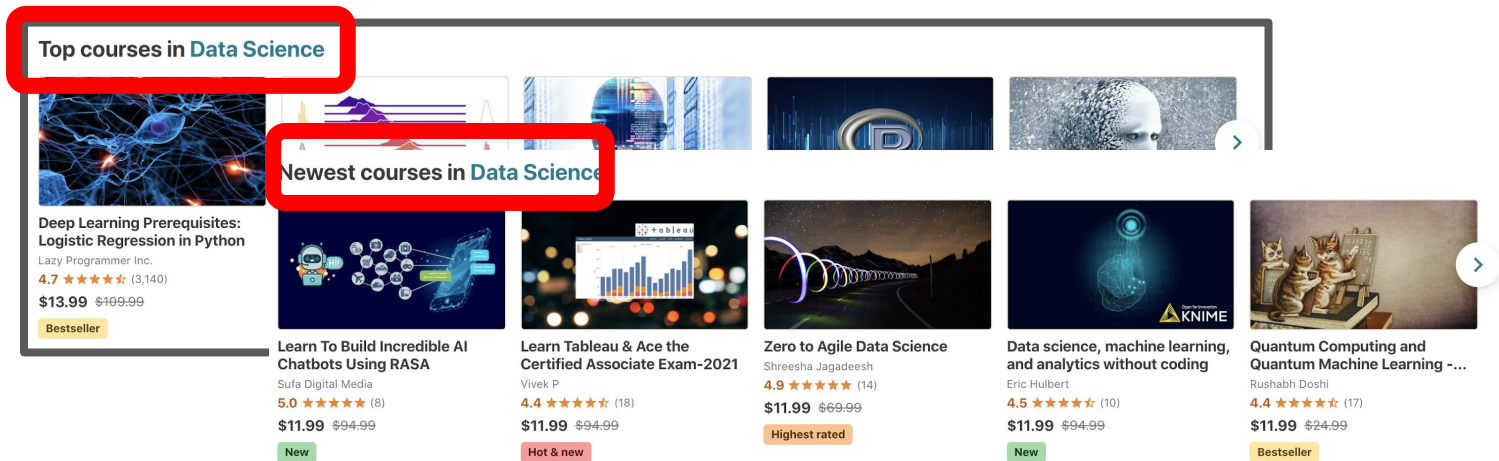
간단한 소개와 실습

## 인기도 기반 추천 유닛 개발 (1)

- Cold Start 이슈가 존재하지 않음
- 인기도의 기준
  - 평점? 매출? 최다 판매?
- 사용자 정보에 따라 확장 가능
  - 서울 지역 인기 아이템 추천
- 단 개인화되어 있지는 않음 (어느 정도는 가능)

## 인기도 기반 추천 유닛 개발 (2)

- 아이템의 분류 체계 정보 존재 여부에 따라 쉽게 확장 가능
  - 특정 카테고리에서의 인기 아이템 추천
  - 분류체계를 갖는 것이 여러모로 유리!!!
- 인기를 다른 기준으로 바꿔 다양한 추천 유닛 생성 가능
  - 예) 인기도 -> 최신성



## 기타 Cold Start 이슈가 없는 추천 유닛

- 현재 사용자들이 구매한 아이템
- 현재 사용자들이 보고 있는 아이템 (영화, 강좌 등등)

### Students are viewing



#### Graphic Design Masterclass - Learn GREAT Design

Lindsay Marsh

4.7 ★★★★★ (22,768)

\$20.99 \$159.99

Bestseller



#### Photography Masterclass: A Complete Guide to Photography

Phil Ebner, William Carnahan, Video Scho...

4.7 ★★★★★ (47,930)

\$19.99 \$149.99

Bestseller



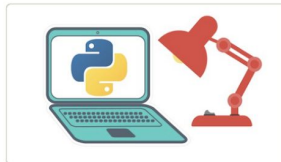
#### The Complete 2021 Web Development Bootcamp

Dr. Angela Yu

4.7 ★★★★★ (117,982)

\$16.99 \$129.99

Bestseller



#### 2021 Complete Python Bootcamp From Zero to Hero i...

Jose Portilla

4.6 ★★★★★ (349,464)

\$17.99 \$139.99

Bestseller



#### The Web Developer Bootcamp 2021

Colt Steele

4.7 ★★★★★ (197,213)

\$18.99 \$139.99

## 실습

- 영화 추천 데이터를 가지고 인기도 기반 추천 엔진 개발
  - TMDB 데이터셋을 사용
- 인기도의 경우 다음 카테고리를 지원
  - 전체 인기도
  - 장르 내 인기도 (일종의 분류 체계)
- 구글 Colab 링크:
  - <https://colab.research.google.com/drive/1K6O9YnHvNGqhm5flPI2tifEVhmMVZDp0?usp=sharing>

# 유사도 측정 방식

코사인 유사도와 피어슨 유사도 설명

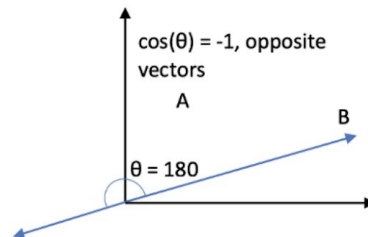
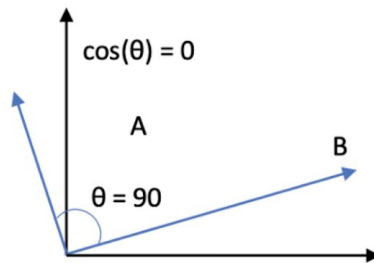
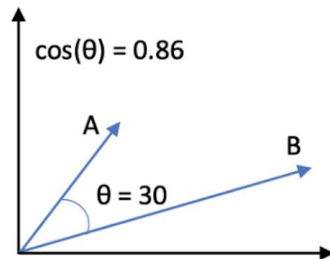
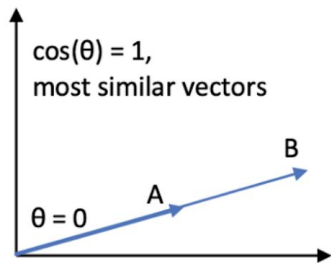


## 알고리즘간 유사도 측정의 차이점

- 협업 필터링에서는 사용자와 아이템을 **평점**으로 표현하고 유사도 측정
- 콘텐츠 기반 추천에서 (보통 텍스트로 구성된) 프로파일을 만들어서 벡터로 변환
  - **Cold Start** 이슈가 없음
  - 예) 옷: 모양, 색상, ...
  - 예) 영화: 제목, 감독, 배우, 장르, ...
  - **Bag of Words, TF-IDF** 등등으로 표현

## 다양한 유사도 측정 알고리즘 (1)

- 벡터들간의 유사도를 판단하는 방법
- 코사인 유사도란?
  - N차원 공간에 있는 두 개의 벡터간의 각도(원점에서)를 보고 유사도를 판단하는 기준
  - Scikit-Learn의 `pair_wise.cosine_similarity` 모듈 사용



소스:

<https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db>

## 다양한 유사도 측정 알고리즘 (2)

- 대표적인 것은 코사인 유사도와 피어슨 유사도
  - 평점처럼 방향 뿐만 아니라 벡터 크기의 정규화가 중요하면 피어슨 유사도를 사용
    - 피어슨 유사도는 코사인 유사도의 개선 버전

코사인 유사도 계산식

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

## 다양한 유사도 측정 알고리즘 (3)

- 피어슨 유사도란?
  - 먼저 벡터 **A**와 **B**의 값들을 보정
    - 각 벡터내 셀들의 평균값을 구한 뒤 평균값을 각 셀에서 빼줌
    - 예를 들어  $A = \{3, 4, 5\}$ 라면 평균값은 4가 되고 보정 후에는  $\{-1, 0, 1\}$ 이 됨
  - 그 이후 계산은 코사인 유사도와 동일
    - 이를 중앙 코사인 유사도 혹은 보정된 코사인 유사도라 부르기도 함
- 피어슨 유사도의 장점
  - 모든 벡터가 원점을 중심으로 이동되고 벡터간의 비교가 더 쉬워짐
    - 평점이란 관점에서는 까다로운 사용자와 아닌 사용자를 정규화하는 효과를 가져옴

# TF-IDF 소개와 실습

문서간 유사도 측정에 사용되는 TF-IDF에 대해 알아보자

## 텍스트를 행렬(벡터)로 표현하는 방법

- 텍스트 문서를 행렬로 표현하는 방법은 여러 가지가 존재
  - 기본적으로 일단 단어를 행렬의 차원으로 표현해야함
  - **Bag of Words** 방식은 문서들에 나타나는 단어수가 **N**개이면 **N**차원으로 문서 표현
    - 딥러닝 워드임베딩 사용시 차원수도 축소. 공간상에서 비슷한 단어들끼리 가깝게 위치
- 원핫 인코딩 + **Bag of Words** (카운트)
  - 단어의 수를 카운트해서 표현
- 원핫 인코딩 + **Bag of Words** (TF-IDF)
  - 단어의 값을 TF-IDF 알고리즘으로 계산된 값으로 표현

## 텍스트를 행렬(벡터)로 표현하는 방법: Bag of Words 예제

- 원핫 인코딩 + Bag of Words (카운트)
  - 단어의 수를 카운트해서 표현

Doc1: deep learning is a machine learning -> deep (1), learning (2), machine (1)

Doc2: machine learning is an AI -> machine (1), learning (1), AI (1)

4개의 단어가 존재함으로 4차원으로 표현 (One-Hot Encoding)

deep [ 1, 0, 0, 0 ], learning [ 0, 1, 0, 0 ], machine [ 0, 0, 1, 0 ], AI [ 0, 0, 0, 1 ]

Doc1 -> [ 1, 2, 1, 0 ]

Doc2 -> [ 0, 1, 1, 1 ]

**Stopwords:** 너무 많이 나와서  
제외하는 단어들. 영어라면 it, is, a  
등등

## 원핫 인코딩 + Bag of Words (카운트)

- 먼저 **stopword** 제외 (the, is, in, we, can, see)
- 그 뒤 단어수 계산 -> 총 5개 sky, blue, sun, bright, shining
- 단어별로 차원을 배정: sky (1), blue (2), sun (3), bright (4), shining (5)

text = [

'The sky is blue.',        # 'sky', 'blue'

'The sun is bright.',      # 'sun', 'bright'

'The sun in the sky is bright',   # 'sun', 'sky', 'bright'

'We can see the shining sun, the bright sun.' # 'shining', 'sun', 'bright'

]

	sky	blue	sun	bright	shining
doc1	1	1	0	0	0
doc2	0	0	1	1	0
doc3	1	0	1	1	0
doc4	0	0	2	1	1



## CountVectorizer 소개

- 앞서 Bag of Words 카운팅 방식을 구현한 모듈
- 이렇게 벡터로 표현이 되면 문서들간의 유사도 측정이 가능

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
text = [  
    'The sky is blue.',  
    'The sun is bright.',  
    'The sun in the sky is bright',  
    'We can see the shining sun, the bright sun.'  
]
```

```
countvectorizer = CountVectorizer(analyzer= 'word', stop_words='english')
```

```
count_wm = countvectorizer.fit_transform(text)
```

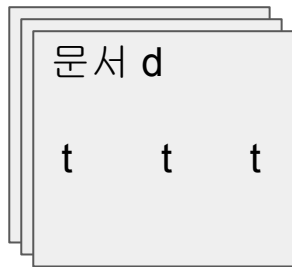
	blue	bright	shining	sky	sun
Doc1	1	0	0	1	0
Doc2	0	1	0	0	1
Doc3	0	1	0	1	1
Doc4	0	1	1	0	2

## TF-IDF 소개

- 앞서 카운트 방식은 자주 나오는 단어가 높은 가중치를 갖게 됨
- TF-IDF의 기본적인 아이디어
  - 한 문서에서 중요한 단어를 카운트가 아닌 문서군 전체를 보고 판단하자
  - 어떤 단어가 한 문서에서 자주 나오면 중요하지만 이 단어가 다른 문서들에서는 자주 나오지 않는다면 더 중요

단어  $t$ 의 문서  $d$ 에서의 점수:  $TF-IDF = TF(t, d) * IDF(t)$

- $TF(t, d)$ : 단어  $t$ 가 문서  $d$ 에서 몇번 나왔나?
- $DF(t)$ : 단어  $t$ 가 전체 문서군에서 몇번 나왔나?
- $IDF(t)$ : 앞서  $DF(t)$ 의 역비율  $= 1/DF(t)$ 
  - 단어가  $t$ 가 전체 문서들중 (총  $N$ 개) 몇개의 문서에서 나왔는지? 이 비율을 역으로 계산한 것이  $IDF$
  - $\ln(N/DF)$ :  $N$ 은 총 문서수를 나타내고  $DF$ 는 단어가 나온 문서를 말한다



## 원핫 인코딩 + Bag of Words (TF-IDF)

- 카운트 기반 Bag of Words랑 동일한데 카운트 대신에 TF-IDF 점수를 사용

$$\text{TF-IDF} = \text{TF}(t, d) * \text{IDF}(t)$$

- TF(t, d): 단어 t의 문서 d에서 카운트
- IDF(t): 역문서 비율. 단어가 t가 전체 문서들중 (총 N개) 몇 개의 문서에서 나타났는지 비율을 역으로 한 값
  - $\ln(N/DF)$ 
    - N은 총 문서수. DF는 단어가 나온 문서의 수를 말한다.
  - 예를 들어 “sun”의 doc4에서의 TF-IDF 점수는

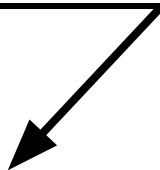
	sky	blue	sun	bright	shining
i. N의 값은 4. “sun”이 나온 문서의 수는 3. Doc4에서 “sun”의 카운트는 2.					
ii. 역문서 점수는 $2 * \ln(4/3) = 0.576$ 과 같다.					
doc1	1, $1 * \log(4/2)$	1, $1 * \log(4/1)$	0, 0	0, 0	0, 0
doc2	0, 0	0, 0	1, $1 * \log(4/3)$	1, $1 * \log(4/3)$	0, 0
doc3	1, $1 * \log(4/2)$	0, 0	1, $1 * \log(4/3)$	1, $1 * \log(4/3)$	0, 0
doc4	0, 0	0, 0	2, $2 * \log(4/3)$	1, $1 * \log(4/3)$	1, $1 * \log(4/1)$

## TfidfVectorizer 소개

```
from sklearn.feature_extraction.text import TfidfVectorizer
text = [
    'The sky is blue.',
    'The sun is bright.',
    'The sun in the sky is bright',
    'We can see the shining sun, the bright sun.'
]
tfidfvectorizer = TfidfVectorizer(analyzer='word', stop_words='english', norm='l2')
tfidf_wm = tfidfvectorizer.fit_transform(text)
```

L2 normalization:

- 벡터를 단위 벡터로 만듦
- 길이가 1인 벡터로 만드는 것



scikit-learn의 TF-IDF 공식은 조금 다름

$$\text{IDF} = \ln(N+1)/(DF+1) + 1$$

왜 다를까? 힌트  $\ln 1$ 은 0, 0으로 나누면

에러

	blue	bright	shining	sky	sun
Doc1	0.785288	0.000000	0.000000	0.619130	0.000000
Doc2	0.000000	0.707107	0.000000	0.000000	0.707107
Doc3	0.000000	0.532570	0.000000	0.657829	0.532570
Doc4	0.000000	0.366260	0.573818	0.000000	0.732521

## TF-IDF로 만든 문서간 유사도 계산


- 앞서 만든 문서 벡터는 `tfidf_wm` 변수에 있음

```
from sklearn.metrics.pairwise import cosine_similarity
```

```
cosine_similarities = cosine_similarity(tfidf_wm)
```

```
print(cosine_similarities)
```

	<b>Doc1</b>	<b>Doc2</b>	<b>Doc3</b>	<b>Doc4</b>
<b>Doc1</b>	1.000000	0.000000	0.407282	0.000000
<b>Doc2</b>	0.000000	1.000000	0.753167	0.776956
<b>Doc3</b>	0.407282	0.753167	1.000000	0.585177
<b>Doc4</b>	0.000000	0.776956	0.585177	1.000000



## TF-IDF의 문제점

- 정확하게 동일한 단어가 나와야 유사도 계산이 이뤄짐
  - 동의어 처리가 안됨
- 단어의 수가 늘어나고 아이템의 수가 늘어나면 계산이 오래 걸림
- 결국은 워드 임베딩을 사용하는 것이 더 좋음
  - 아니면 LSA (Latent Semantic Analysis)와 같은 차원 축소 방식 사용

## CountVectorizer/TfidfVectorizer/코사인 유사도 실습

- 앞서 설명한 개념들을 실제 코드로 실습
- 이를 바탕으로 뒤에서 컨텐츠 기반 추천 엔진 개발
- 실습 링크
  - <https://colab.research.google.com/drive/1WRd0uhPSwLVuQk998teEaziCHesMNa-wD?usp=sharing>

# TF-IDF를 이용한 콘텐츠 기반 추천과 실습

앞서 TMDB 데이터셋을 이용해 비슷한 영화 추천 구현



## 실습

- 앞서 인기 영화 추천 실습에서 사용했던 **TMDB** 데이터를 재사용
- 콘텐츠 기반 추천 엔진 개발
  - TfidfVectorizer와 cosine\_similarity를 사용
- 구글 colab 링크:
  - <https://colab.research.google.com/drive/1xbFjBvzgiDpPwBwKiaMw0ByWos12EAiD?usp=sharing>

- 넷플릭스 프라이즈 소개
- 유데미 추천 방식 소개
- 인기도 기반 추천 방식 소개
- 유사도 계산 (코사인과 피어슨)
- TF-IDF와 이를 기반으로한 컨텐츠 기반 추천 엔진 실습



UpZen

keeyong@gmail.com