December 10, 2025

▣ **강의명:** CSCI E-89B: 자연어 처리 입문

▣ **주차:** Lecture 09

▣ **교수명:** Dmitry Kurochkin

▣ **목적:** Lecture 09의 핵심 개념 학습

Contents

1 개요 (Overview)

□ 핵심 요약

이 문서는 자연어 처리(NLP) 작업, 특히 텍스트 분류를 수행하기 위한 **고전적 기계학습(Classical Machine Learning) 모델**들을 다룹니다.
데이터가 아주 많지 않을 때(e.g., 수천 수만 건) 딥러닝보다 효과적일 수 있는 **Naive Bayes, KNN, Logistic Regression, SVM, Random Forest**의 핵심 원리와 장단점을 초심자의 시선에서 설명합니다.
특히, 현실의 **불균형 데이터(imbalanced data)**를 다룰 때 단순 '정확도'의 함정을 피하고, **K-겹 교차 검증(K-Fold CV), 파이프라인(Pipeline), 혼동 행렬(Confusion Matrix)**을 활용하여 모델을 올바르게 평가하고 최적화하는 실전 방법론에 초점을 맞춥니다.

2 학습 로드맵 및 주요 용어 정리

2.1 학습 로드맵

이 노트를 효과적으로 학습하기 위한 권장 순서입니다.

1. **텍스트의 숫자 변환**: 왜 TF-IDF가 필요한지 이해합니다.
2. **모델 검증의 중요성**: 왜 단순 8:2 분할 대신 K-겹 교차 검증(K-Fold CV)을 쓰는지 이해합니다.
3. **평가 지표의 함정**: 왜 '정확도(Accuracy)'가 아닌 '정밀도(Precision)'와 '재현율(Recall)'이 중요한지 이해합니다.
4. **개별 모델 학습**: 각 모델(Naive Bayes, KNN, SVM 등)의 기본 작동 원리를 비유를 통해 이해합니다.
5. **실습 사례 분석**: BBC 뉴스 분류 예제에서 어떤 모델이 왜 더 좋은 성능을 보였는지 확인합니다.

2.2 주요 용어 정리표

이 문서에서 자주 등장하는 핵심 용어들입니다.

Table 1: 핵심 용어 정리

용어 (한글)	용어 (원어)	쉬운 설명	비고
TF-IDF	Term Frequency-Inverse Document Frequency	"이 문서에선 자주 나오지만, 다른 데선 잘 안 나오는 단어"에 높은 점수를 줌	텍스트의 숫자화
K-겹 교차 검증	K-Fold Cross-Validation	데이터를 K개로 쪼개서, 한 조각씩 돌아가며 테스트하고 평균내는 검증법	데이터가 적을 때 유용
파이프라인	Pipeline	데이터 전처리(e.g., TF-IDF)와 모델 학습을 묶어주는 통로	데이터 누수 방지
혼동 행렬	Confusion Matrix	모델이 무엇을 맞혔고(TP, TN), 무엇을 틀렸는지(FP, FN) 보여주는 표	평가의 시작
정확도	Accuracy	(TP + TN) / 전체. **불균형 데이터에선 함정이 될 수 있음**	100개 중 99개 맞힘
정밀도	Precision	TP / (TP + FP). "모델이 '양성'이라고 한 것 중, **진짜 '양성'**일 확률"	스팸 필터 (신뢰도)
재현율 / 민감도	Recall / Sensitivity	TP / (TP + FN). "**실제 '양성'**인 것 중, 모델이 **찾아낸** 비율"	암 진단 (누락 방지)
특이도	Specificity	TN / (TN + FP). "**실제 '음성'**인 것 중, 모델이 **'음성'**이라고 맞춘 비율"	–
하이퍼파라미터	Hyperparameter	모델이 학습하기 전에 **사람이 직접 설정**해줘야 하는 값	e.g., KNN의 'k' 값
GridSearchCV	Grid Search Cross-Validation	하이퍼파라미터 후보들을 **모두 조합**해보고 CV로 최고를 찾는 도구	자동 최적화

3 왜 딥러닝 대신 고전적 모델을 사용하나요?

최근 NLP 분야는 딥러닝(Deep Learning) 모델들이 주도하고 있지만, 고전적 기계학습 모델은 여전히 중요하며 특정 상황에서 더 유리합니다.

- **데이터 크기**: 딥러닝 모델은 성능을 내기 위해 **아주 많은 데이터**가 필요합니다. 데이터가 수천 수만 건 정도로 비교적 적다면, 딥러닝 모델은 과적합(Overfitting)되기 쉬운 반면, 고전적 모델들(특히 SVM)이 더 안정적이고 우수한 성능을 보일 때가 많습니다.
- **학습 속도 및 비용**: 고전적 모델은 딥러닝 모델보다 훨씬 **빠르게 훈련**됩니다. 복잡한 GPU 환경 설정 없이 일반 CPU로도 충분히 빠릅니다.
- **해석 용이성**: Logistic Regression이나 Decision Tree 같은 모델은 어떤 단어(Feature)가 분류에 큰 영향을 미쳤는지 비교적 쉽게 해석할 수 있습니다. (물론 SVM(커널 사용 시)이나 Random Forest는 딥러닝처럼 해석이 어려운 '블랙박스'에 가깝습니다.)

4 핵심 절차 1: 텍스트 벡터화 (TF-IDF)

기계학습 모델은 '뉴스 기사'라는 텍스트 자체를 이해할 수 없습니다. 오직 숫자, 즉 **벡터(Vector)**만을 입력으로 받습니다. 따라서 텍스트를 숫자로 변환하는 과정이 반드시 필요합니다.

가장 널리 쓰이는 방법이 **TF-IDF**입니다.

- **TF (Term Frequency, 단어 빈도)**: 한 문서 내에서 특정 단어가 얼마나 **자주** 등장했는지를 나타냅니다. (e.g., 이 문서에서 '모델'이라는 단어의 TF 값은 높습니다.)
- **IDF (Inverse Document Frequency, 역문서 빈도)**: 특정 단어가 전체 문서 집합에서 얼마나 **희귀**한지를 나타냅니다. 모든 문서에 다 나오는 단어(e.g., 'the', 'a', '이다')는 희귀성이 낮아 IDF 값이 0에 가깝습니다. 반면, 특정 주제의 문서에서만 나오는 단어(e.g., 'SVM', '커널')는 희귀성이 높아 IDF 값이 큽니다.

TF-IDF 값 = TF × IDF

결과적으로, 해당 문서에서 **자주 등장**하면서 (높은 TF) 다른 문서에서는 **잘 안 나오는** (높은 IDF) 단어일수록 그 문서를 대표하는 **중요한 키워드**로 인식되어 높은 TF-IDF 점수를 받게 됩니다.

5 핵심 절차 2: 모델 훈련 및 검증

5.1 단순 분할 vs. K-겹 교차 검증 (K-Fold CV)

문제점: 2,225개의 뉴스 기사 데이터가 있다고 가정해봅시다. (e.g., 훈련 80%, 테스트 20%로 분할)

이 방식은 **어떤** 데이터를 테스트용으로 뽑았는지에 따라 모델 성능이 **우연히** 좋거나 나쁘게 나올 수 있습니다. (결과가 불안정함) 또한, 전체 데이터의 20%(약 445개)를 테스트용으로 "낭비"하게 됩니다. 데이터가 많지 않을수록 이 "낭비"는 뼈아픕니다.

□ 예제:

해결책: K-겹 교차 검증 (K-Fold Cross-Validation, k=5 예시)
"낭비"되는 데이터 없이, 모든 데이터를 훈련과 검증에 골고루 사용하는 방법입니다.

1. 전체 데이터를 K개(e.g., 5개)의 "폴드(Fold)" 또는 "조각"으로 나눕니다.
2. **실행 1:** 1 4번 조각으로 **훈련** → 5번 조각으로 **테스트** (성능 기록)
3. **실행 2:** 1 3, 5번 조각으로 **훈련** → 4번 조각으로 **테스트** (성능 기록)
4. **실행 3:** 1 2, 4 5번 조각으로 **훈련** → 3번 조각으로 **테스트** (성능 기록)
5. **실행 4:** 1, 3 5번 조각으로 **훈련** → 2번 조각으로 **테스트** (성능 기록)
6. **실행 5:** 2 5번 조각으로 **훈련** → 1번 조각으로 **테스트** (성능 기록)

최종 결과: 5번의 테스트 성능 점수(e.g., [95, 97, 96, 98, 96])의 **평균**을 냅니다. 이 평균값은 단 한 번의 8:2 분할보다 훨씬 **안정적이고 신뢰할 수 있는** 모델 성능 추정치입니다.

5.2 `Pipeline`의 중요성: 데이터 누수(Data Leakage) 방지

K-Fold CV의 치명적 함정: 데이터 누수 (Data Leakage)

만약 K-Fold CV를 **하기 전에, 전체 2,225개**의 데이터로 `TfidfVectorizer`를 `fit` (학습) 시켰다고 가정해봅시다.

이는 훈련(1 4번 조각)을 할 때, **미리 테스트 데이터(5번 조각)의 정보(IDF 값)를 엿본 셈**이 됩니다. 이는 현실에서 일어날 수 없는 일이며, 모델 성능이 비정상적으로 높게 나오는 원인이 됩니다. (반칙!)

올바른 절차: 매 실행마다(총 5번), **오직 훈련용 조각(e.g., 1 4번)만**으로 `TfidfVectorizer`를 **새롭게** `fit` 하고, 그것으로 테스트 조각(e.g., 5번)을 `transform` 해야 합니다.

이 복잡하고 반복적인 "올바른 절차"를 아주 쉽게 자동화해주는 도구가 바로 `sklearn`의 **`Pipeline`**입니다.

`Pipeline`은 (1) TF-IDF 벡터화, (2) 분류 모델 학습 이 두 단계를 하나의 "파이프"로 묶어줍니다. 이 `Pipeline` 객체를 `cross_val_score` 같은 교차 검증 함수에 전달하면, 함수가 **알아서** 매 실행마다 훈련용 조각으로만 벡터화(fit)를 수행하여 데이터 누수를 완벽하게 방지합니다.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score

# 1. Pipeline 정의: 단계를2 하나로묶음
```

```
# 'vec': TF-IDF 벡터화단계
# 'clf': Naive Bayes 분류기단계
text_clf_pipeline = Pipeline([
    ('vec', TfidfVectorizer()),
    ('clf', MultinomialNB()),
])

# 2. 을Pipeline 교차검증기에전달
# X_data: 전체텍스트데이터   (e.g., 개2225 기사)
# y_data: 전체레이블  (e.g., 개2225 0 또는 1)
# cv=5: 겹5- 교차검증을수행
# 이과정에서데이터누수없이번의      5 fit/이transform 자동으로일어남
scores = cross_val_score(text_clf_pipeline, X_data, y_data, cv=5, scoring='
    accuracy')

print(f"교차 검증평균정확도  : {scores.mean():.4f}")
```

Listing 1: sklearn의 Pipeline을 사용한 교차 검증 예시

5.3 GridSearchCV를 이용한 하이퍼파라미터 최적화

- **하이퍼파라미터란?** 모델이 데이터로부터 "학습"하는 값(e.g., Logistic Regression의 가중치 w)이 아니라, 우리가 모델에게 **미리 알려줘야 하는 설정값**입니다. (e.g., KNN의 k 값, SVM의 C 값과 'kernel' 종류) 이 설정값에 따라 모델 성능이 크게 달라집니다.
- **GridSearchCV란?** "Grid Search" + "Cross-Validation"의 합성어입니다. 우리가 시도해보고 싶은 하이퍼파라미터 후보들을 **모두 조합**하여 (Grid Search), 각 조합마다 **K-Fold CV**를 수행하여 가장 성능이 좋았던 **최고의 조합**을 찾아줍니다.

□ 예제:

KNN 모델의 최적 k 찾기 예시
GridSearchCV에게 "k 값으로 1, 3, 5, 10, 15, 19를 테스트해보고, 각 k 마다 5-Fold CV를 돌려서 평균 점수가 가장 높은 k를 알려줘"라고 명령하는 것과 같습니다.

□ 예제:

SVM 모델의 최적 C와 'kernel' 찾기 예시
후보군: 'C = [0.1, 1, 10]', 'kernel = ['linear', 'rbf']' GridSearchCV는 아래 6가지 조합을 모두 테스트합니다.

- (C=0.1, kernel='linear') → 5-Fold CV → 평균 점수 1
- (C=0.1, kernel='rbf') → 5-Fold CV → 평균 점수 2
- (C=1, kernel='linear') → 5-Fold CV → 평균 점수 3
- (C=1, kernel='rbf') → 5-Fold CV → 평균 점수 4
- (C=10, kernel='linear') → 5-Fold CV → **평균 점수 5 (최고!)**
- (C=10, kernel='rbf') → 5-Fold CV → 평균 점수 6

최종 결과: "최고의 조합은 'C=10', 'kernel='linear'' 입니다."

최종 모델 훈련

`GridSearchCV`는 최적의 하이퍼파라미터를 **찾아주는** 도구입니다. 최고의 조합(e.g., 'C=10, kernel='linear' ')을 찾았다면, 그 설정으로 **전체 훈련 데이터**를 다시 학습시켜 **단 하나의 최종 모델**을 만들어야 합니다.

6 핵심 절차 3: 불균형 데이터 평가하기

6.1 정확도(Accuracy)의 함정

문제 상황: "테크" 뉴스(401개)와 "비-테크" 뉴스(1824개)를 분류하는 문제. 데이터 비율이 약 1:4.5로 **불균형(Imbalanced)**합니다.

함정: 만약 어떤 모델이 **모든 기사를 무조건 "비-테크"라고만 예측**한다면, 이 "멍청한" 모델은 1824개는 맞히고 401개는 틀리게 됩니다. 이 모델의 **정확도(Accuracy)**는 $1824/(1824+401) = 81.9\%$ 입니다.

81.9%라는 높은 정확도 숫자만 보면 이 모델이 꽤 쓸만하다고 **오해**할 수 있지만, 우리의 원래 목적인 "테크" 기사를 **단 하나도** 찾아내지 못하는 완전히 쓸모없는 모델입니다.

6.2 해결책: 혼동 행렬 (Confusion Matrix)

불균형 데이터를 평가할 때는, 모델이 "무엇을" 맞혔고 "무엇을" 틀렸는지 세부적으로 파악해야 합니다. 이때 사용하는 것이 **혼동 행렬**입니다.

("테크" 뉴스를 '양성(Positive)', "비-테크" 뉴스를 '음성(Negative)'이라고 가정)

Table 2: 혼동 행렬 (*Confusion Matrix*)

		모델의 예측 (Predicted)	
		'테크' (Positive)	'비-테크' (Negative)
실제 값 (Actual)	'테크' (Positive)	**TP (True Positive)** (진짜 '테크'를 '테크'로 맞힘)	**FN (False Negative)** (진짜 '테크'를 '비-테크'로 놓침)
	'비-테크' (Negative)	**FP (False Positive)** ('비-테크'를 '테크'로 잘못 예측)	**TN (True Negative)** ('비-테크'를 '비-테크'로 맞힘)

6.3 핵심 평가 지표: 정밀도, 재현율, 특이도

혼동 행렬의 4가지 값(TP, TN, FP, FN)을 조합하여 정확도보다 훨씬 유용한 3가지 핵심 지표를 만듭니다.

Table 3: 불균형 데이터의 핵심 평가 지표

지표	공식	**직관적 의미 (일상 언어 번역)**
정밀도 (Precision)	$\frac{TP}{TP+FP}$	"모델이 '테크'라고 예측한 것들 중, **진짜 '테크'**일 확률" **"이 모델의 예측은 얼마나 믿을만한가?"** (신뢰도)
재현율 (Recall) (민감도, Sensitivity)	$\frac{TP}{TP+FN}$	"**실제 '테크'** 기사들 중, 모델이 **얼마나 많이 찾아냈는가?**" **"이 모델이 놓친 것은 얼마나 적은가?"** (누락도)
특이도 (Specificity)	$\frac{TN}{TN+FP}$	"**실제 '비-테크'** 기사들 중, 모델이 **'비-테크'라고 맞춘** 비율" (재현율의 '비-테크' 버전)

□ 예제:

정밀도 vs. 재현율의 트레이드오프(Trade-off)

우리는 두 지표가 모두 높기를 원하지만, 현실에서는 종종 한쪽이 높아지면 다른 쪽이 낮아지는 **트레이드오프** 관계가 발생합니다.

- **상황 1: 스팸 메일 필터 (정밀도가 중요)** 필터가 **아주 확실한** 스팸(e.g., 99.9% 확신)만 걸러낸다고 가정합시다.
  - **정밀도 (Precision) = 매우 높음**: 필터가 "스팸"이라고 한 것은 100% 스팸입니다. (신뢰도 높음)
  - **재현율 (Recall) = 낮음**: 애매한 스팸(e.g., 80% 확신)은 **놓치고** 받은 편지함으로 통과시킵니다.
  - **결론**: 중요한 메일이 스팸함으로 가는 것(FP)을 막는 것이, 스팸을 몇 개 놓치는 것(FN)보다 중요합니다.
- **상황 2: 암 진단 모델 (재현율이 중요)** 모델이 암일 **가능성이 조금이라도** 있으면 "양성"이라고 예측한다고 가정합시다.
  - **재현율 (Recall) = 매우 높음**: **실제 암 환자(TP)를 놓치는 일(FN)이 거의 없습니다.**
  - **정밀도 (Precision) = 낮음**: "양성" 예측(TP+FP) 중에 **정상인(FP)이 많이 섞여있습니다.**
  - **결론**: 정상인을 암으로 오진(FP)하여 추가 검사를 하는 한이 있더라도, 실제 암 환자를 정상이라고 놓치는(FN) 치명적인 실수는 절대 안 됩니다.

"테크" 뉴스 분류 문제는 정답이 없습니다. "나는 테크 기사를 **하나도 놓치기 싫어!**"라면 **재현율**이 높은 모델을, "내가 추천받은 기사는 **무조건 테크 기사여야 해!**"라면 **정밀도**가 높은 모델을 선택해야 합니다.

7 고전적 분류 모델 상세

7.1 Naive Bayes (나이브 베이즈)

- **한 줄 요약**: 베이즈 정리를 기반으로, "모든 단어는 서로 독립적"이라고 **순진하게(Naive)** 가정하는 분류기.
- **직관적 예시 (스팸 필터)**: 어떤 메일에 'free'와 'Viagra'라는 단어가 동시에 등장했습니다.
  - **현실**: 'free'와 'Viagra'는 스팸 메일에서 **함께** 등장할 확률이 높습니다. (서로 의존적)
  - **Naive Bayes의 가정**: "나는 똑똑하지 않아서(Naive) 그런 관계는 모르겠고, 'free'가 등장할 확률과 'Viagra'가 등장할 확률을 그냥 **곱할래**." 즉, $P(\text{'free' and 'Viagra'}|\text{스팸}) = P(\text{'free'}|\text{스팸}) \times P(\text{'Viagra'}|\text{스팸})$ 이라고 가정합니다.

  이 가정은 명백히 틀렸지만, 놀랍게도 스팸 필터링 같은 많은 문제에서 빠르고 준수한 성능을 보여줍니다.
- **기술적 설명**: 베이즈 정리에 따라, 우리는 $P(C_k|\mathbf{x})$ (특징 $\mathbf{x}$가 주어졌을 때 클래스 C_k 일 확률)를 최대화하는 클래스 C_k를 찾습니다.

$$\hat{C} = \arg\max_{C_k} P(C_k|\mathbf{x}) = \arg\max_{C_k} \frac{P(\mathbf{x}|C_k)P(C_k)}{P(\mathbf{x})}$$

  여기서 $\mathbf{x}$는 (TF-IDF 벡터) $(x_1, x_2, ..., x_n)$입니다. $P(\mathbf{x})$는 모든 클래스에서 동일하므로 무시하고, $P(C_k)$는 단순히 훈련 데이터의 클래스 비율(사전 확률)입니다.
  핵심은 $P(\mathbf{x}|C_k)$인데, **나이브 가정**에 따라 다음과 같이 계산합니다.

$$P(\mathbf{x}|C_k) = P(x_1, ..., x_n|C_k) \approx \prod_{i=1}^{n} P(x_i|C_k)$$

  실제로는 값들이 너무 작아져서 0이 되는 것을 막기 위해(numerical underflow), 로그(log)를 씌워서 곱셈을 덧셈으로 바꿔 계산합니다.

$$\log P(C_k|\mathbf{x}) \propto \log P(C_k) + \sum_{i=1}^{n} \log P(x_i|C_k)$$

- **장점**: 매우 빠르고, 구현이 간단하며, 적은 데이터로도 잘 작동합니다. 훌륭한 "베이스라인" 모델입니다.
- **단점**: "모든 특징이 독립"이라는 가정이 현실과 너무 다르면 성능이 떨어집니다.

7.2 k-Nearest Neighbors (KNN, k-최근접 이웃)

- **한 줄 요약**: 별다른 "학습" 없이, 그냥 데이터를 다 외워버린 다음, 새로운 데이터가 들어오면 **가장 가까운 k개의 이웃**을 보고 **다수결 투표**로 결정하는 모델.
- **직관적 예시 ("동네 투표")** 새로 이사 온 집(새 데이터)이 'A동'인지 'B동'인지 분류해야 합니다.
  1. k값을 정합니다. (e.g., $k = 5$)
  2. 새 집에서 **가장 가까운 집 5개(이웃)**를 찾습니다.
  3. 5개 이웃을 보니, 3채는 'A동', 2채는 'B동'이었습니다.
  4. **다수결의 원칙**에 따라, "새 집은 **'A동'**일 것이다"라고 분류합니다.

KNN은 이 과정이 전부입니다. 별도의 "훈련" 단계가 없고, 데이터 자체를 모델로 사용합니다. (Instance-based Learning)

- **기술적 설명**:
  - **하이퍼파라미터 1: k 값**: k가 너무 작으면(e.g., $k = 1$) 이웃 한 명의 "이상한" 의견(outlier)에 휘둘리기 쉽고 (과적합), k가 너무 크면 너무 먼 동네 의견까지 들어서 분류 경계가 둔해집니다 (과소적합). 보통 3, 5, 7 등 **홀수**를 사용합니다. (동점 투표를 피하기 위해)
  - **하이퍼파라미터 2: 거리 측정 (Distance Metric)**: "가깝다"를 어떻게 정의할지 정해야 합니다.
    - **유클리드 거리 (Euclidean)**: 두 점 사이의 직선 거리. (가장 일반적)
    - **코사인 유사도 (Cosine Similarity)**: 텍스트 분류에서 자주 사용됨. 두 TF-IDF 벡터가 가리키는 **방향**이 얼마나 유사한지 측정. (문서 길이와 상관없이 주제의 유사성을 봄)
- **장점**: 모델이 매우 단순하고 직관적입니다. 훈련이 필요 없습니다.
- **단점**:
  - **데이터 정규화(Normalization) 필수**: 만약 [나이(1 100), 연봉(1만 100만)]으로 이웃을 찾는다면, '나이' 차이(e.g., 5)는 '연봉' 차이(e.g., 50,000)에 비해 완전히 무시됩니다. 모델은 **오직 연봉**만 보게 됩니다. 따라서 모든 특징(feature)의 범위를 [0, 1] 등으로 **스케일링**하는 작업이 **반드시** 필요합니다.
  - **느린 예측**: 예측 시마다 **모든 훈련 데이터**와의 거리를 계산해야 하므로, 데이터가 수백만 개가 되면 예측이 매우 느려집니다.

7.3 Logistic Regression (로지스틱 회귀)

- **한 줄 요약**: 선형 회귀(직선)의 결과를 **시그모이드(Sigmoid) 함수**에 넣어 0 1 사이의 **"확률"** 값으로 변환하는 분류기.
- **직관적 예시 (단 하나의 뉴런)**: 로지스틱 회귀는 딥러닝의 **뉴런 1개**와 거의 동일합니다.
  1. **입력 (Inputs)**: 각 단어의 TF-IDF 점수들 $(x_1, x_2, ...)$.
  2. **가중치 (Weights)**: 각 단어가 '테크' 분류에 얼마나 중요한지 나타내는 가중치 $(w_1, w_2, ...)$. (e.g., 'apple'의 가중치는 높고, 'sport'의 가중치는 음수일 것)
  3. **선형 결합 (z)**: 모든 (점수 × 가중치)를 더합니다. $\rightarrow z = w_1x_1 + w_2x_2 + ... + b$ (z 값은 $-\infty$ $+\infty$ 사이의 어떤 값이 됩니다.)
  4. **활성화 함수 (Sigmoid)**: z 값을 시그모이드 함수에 넣어 0 1 사이의 값으로 "찌그러뜨립니다". $\rightarrow \sigma(z) = \frac{1}{1+e^{-z}}$ (e.g., $z = 10 \rightarrow \sigma(z) \approx 1.0$, $z = -5 \rightarrow \sigma(z) \approx 0.0$, $z = 0 \rightarrow \sigma(z) = 0.5$)

  이 0 1 사이의 최종 값이 "테크" 기사일 확률(e.g., 0.9)이 되며, 보통 0.5를 기준으로 '테크'(>0.5) / '비-테크'(<0.5)를 결정합니다.
- **기술적 설명**:
  - **선형 결정 경계 (Linear Boundary)**: 로지스틱 회귀는 기본적으로 **직선(또는 초평면)**으로만 데이터를 나눌 수 있습니다.
  - **비선형 문제 해결법**: 만약 데이터가 원형으로 분포(e.g., 중앙은 'A', 바깥은 'B')한다면 직선으로 나눌 수 없습니다. 이때는 **특성 공학(Feature Engineering)**을 통해 해결합니다. 기존 입력 x_1, x_2에 더해 x_1^2, x_2^2 같은 **새로운 특성**을 우리가 **직접** 만들어서 모델에 넣어주면, 모델이 $w_3(x_1^2) + w_4(x_2^2)$ 같은 항을 학습하여 **원형의 경계**를 만들 수 있게 됩니다.
- **장점**: 출력이 0 1 사이의 확률이라 해석하기 좋습니다. 딥러닝의 기초 개념을 이해하는 데 도움이 됩니다.

- **단점**: 기본적으로 선형 분류기이므로, 비선형 경계를 가지는 복잡한 문제는 잘 풀지 못합니다. (SVM의 커널 트릭과 대비됨)

7.4 Support Vector Machines (SVM, 서포트 벡터 머신)

- **한 줄 요약**: 2012년 딥러닝이 부상하기 전까지 "왕좌"에 있던 모델. 두 클래스 사이의 **간격(Margin)을 최대화**하는 결정 경계(선)를 찾는 모델.
- **직관적 예시 ("가장 넓은 도로 찾기")** 서로 다른 두 마을('A' 마을, 'B' 마을) 사이에 국경선을 긋는다고 생각해봅시다.
  - **다른 모델**: 그냥 'A'와 'B'를 나누는 **선(Line)** 하나만 그으려고 합니다.
  - **SVM**: 선 하나가 아니라, 두 마을 사이에 **가장 넓은 중립 도로(Margin)**를 낸다고 생각합니다.

  이때 도로의 **양쪽 경계선**에 정확히 **닿는** 집들(데이터 포인트)이 생기는데, 이 집들이 바로 도로의 위치를 결정하는 가장 중요한 집들, 즉 **서포트 벡터(Support Vectors)**입니다. SVM은 이 "도로 폭(Margin)"을 **최대화**하는 선을 찾습니다.
- **기술적 설명 (단계별 진화)**

  1. 하드 마진 (Hard Margin)
  - **개념**: 단 하나의 데이터도 도로(마진) 안으로 들어오는 것을 허용하지 않는 "완벽한" 도로.
  - **문제 1 (과적합)**: 도로 폭이 오직 양쪽의 가장 가까운 집 2 3개(서포트 벡터)에만 의존합니다. 이 집들이 약간의 노이즈(Outlier)라면, 도로 전체가 심하게 휘어집니다. (과적합)
  - **문제 2 (불가)**: 두 마을의 집들이 약간 섞여있다면(Linearly non-separable), 하드 마진 도로는 아예 만들 수조차 없습니다.

  2. 소프트 마진 (Soft Margin) - 하이퍼파라미터 C
  - **개념**: 현실적인 타협. "약간의 집(데이터)이 도로 안(마진 침범)에 있거나, 심지어 **국경 너머(잘못 분류)**에 있는 것을 **허용**하자. 대신 **벌금(C)**을 내자."
  - **하이퍼파라미터 C (벌금의 세기)**:
  - **C가 크다 (High C)**: **벌금이 아주 비싸다.** → SVM은 벌금을 내기 싫어서 마진을 침범하지 않으려고 **좁은** 도로를 만듭니다. (하드 마진과 유사해짐, 과적합 위험)
  - **C가 작다 (Low C)**: **벌금이 싸다.** → SVM은 "벌금 좀 내지 뭐" 하면서 마진을 침범하는 데이터들을 너그럽게 봐주고, 대신 **도로 폭(마진)을 넓게** 확보합니다. (일반화 성능 향상)
  - **효과**: 소프트 마진 덕분에 데이터가 섞여있어도 분류가 가능해지고, 과적합이 줄어듭니다.

  3. 커널 트릭 (The Kernel Trick) - 'kernel='rbf'' / ''poly''

  □ **예제:**

  문제: 데이터가 ['A' 마을이 'B' 마을을 **원형으로 둘러싼**] 형태입니다. 이건 2D 평면에서는 **절대 직선(도로)으로** 나눌 수 없습니다.
  잘못된 직관: "SVM이 원형 경계선을 그린다."
  올바른 직관 (커널 트릭):
  1. 2D 지도를 3D로 바꿉니다. $z = x^2 + y^2$ (중심에서 먼 만큼 높이 솟아오름) 라는 3번째 차원을 **추가**합니다.
  2. 2D 지도에서는 'A'가 'B'를 둘러쌌지만, 3D 공간에서는 바깥쪽 'B' 마을이 **더 높이 솟아오르고 (높은 z)**, 안쪽 'A' 마을은 **아래 (낮은 z)**에 있게 됩니다.
  3. 이제 이 3D 공간에서는 **단순한 평면(Hyperplane)** 하나로 'A'와 'B'를 완벽하게 분리(e.g., $z = 0.5$ 평면)할 수 있게 됩니다!

  커널(Kernel)이란, $z = x^2 + y^2$ 같은 새로운 차원을 **실제로 계산하지 않고도** (계산량이 너무 많아짐), 마치 그 고차원에서 계산한 것과 **동일한 결과**를 2D에서 **값싸게** 얻어내는 수학적 "트릭"

입니다.
- `kernel='linear'`: 커널 트릭 안 씀. (데이터가 이미 고차원이거나 선형 분리 가능할 때 좋음)
- `kernel='rbf'`: (Radial Basis Function) 데이터를 무한 차원으로 매핑. (가장 범용적이고 강력함)
- `kernel='poly'`: 다항식(x^2, x_1x_2 등) 차원을 추가.

- **장점**: **매우 강력합니다.** 특히 텍스트 데이터(TF-IDF)처럼 **매우 고차원**(수만 차원)인 공간에서는, 데이터가 선형으로 분리 가능할 확률이 높기 때문에 'kernel='linear''인 SVM이 압도적으로 빠르고 좋은 성능을 보일 때가 많습니다. 비선형 문제도 커널 트릭으로 잘 해결합니다.
- **단점**: 모델이 블랙박스에 가깝고, 데이터가 아주 많아지면 훈련 속도가 느려집니다.

7.5 Random Forests (랜덤 포레스트)

- **한 줄 요약**: 과적합되기 쉬운 **결정 트리(Decision Tree)** 모델 **수백 개**를 만들되, 각각을 **일부러 조금씩 다르게** 만들어서(Random), 그 결과들을 **평균(회귀) 또는 투표(분류)**하는 앙상블(Ensemble) 모델.
- **직관적 예시 ("현명한 군중")**
  - **결정 트리 1개**: "한 명의 **천재 전문가**"와 같습니다. 이 전문가는 아주 복잡한 규칙(e.g., "연봉이 5만 이상이고, 나이가 30 미만이며, 거주지가 서울이 아니면...")을 만들어 데이터를 완벽하게 외워버립니다. **문제**: 이 전문가는 훈련 데이터는 100% 맞히지만, 자신의 지식(규칙)에 너무 빠져있어서(과적합) 새로운 데이터를 만나면 엉뚱한 예측을 할 수 있습니다.
  - **랜덤 포레스트**: "수백 명의 **다양한 전문가**로 이루어진 **군중**"과 같습니다. 한 명의 천재보다 **조금 덜 똑똑한 다수의 군중**이 더 현명한 결정을 내린다는 원리입니다. **어떻게 "다양한" 전문가를 만드나요? (이것이 "Random"의 핵심)**
    1. **부트스트랩 (Bootstrapping, 다른 데이터)**: 1000개의 데이터가 있다면, **복원 추출** (샘플 뽑고 다시 집어넣기)로 1000개를 뽑습니다. (어떤 데이터는 2 3번 뽑히고, 어떤 데이터는 아예 안 뽑힘) 이 "조금씩 다른 훈련 세트"를 500개 만들어서 500명의 전문가(트리)에게 각각 나눠줍니다. → 모든 전문가가 **서로 다른 데이터**로 학습합니다.
    2. **특성 랜덤성 (Feature Randomness, 다른 관점)**: 각 전문가(트리)가 결정을 내릴 때(split), **모든 정보(특성)를 보지 못하게 합니다.** (e.g., 전체 특성이 20개라면, "당신은 5개 특성 중에서만 골라서 결정하세요"라고 제한함) → 모든 전문가가 **서로 다른 관점**으로 문제를 보게 됩니다.

    최종 결정: 500명의 전문가가 **다수결 투표**를 합니다. 개별 전문가는 과적합되었을지 몰라도, 그들의 실수가 **서로 다르기 때문에** 투표 과정에서 **상쇄**됩니다.
- **기술적 설명**: 앙상블 기법 중 **배깅(Bagging)** = **B**ootstrap **Agg**regat**ing**에 특성 랜덤성을 추가한 모델입니다. 결정 트리는 데이터를 나눌 때 **불순도(Impurity)**를 가장 낮추는 방향으로 나눕니다. (Gini Index 또는 Entropy 사용) 랜덤 포레스트는 이 과정을 수백 번 반복하여 그 결과를 취합합니다.
- **장점**: **매우 강력하고 안정적입니다.** 데이터 스케일링(정규화)이 필요 없습니다. 과적합에 매우 강건하여, 하이퍼파라미터 튜닝에 크게 신경 쓰지 않아도 (e.g., 기본값 사용) 거의 항상 준수한 성능을 냅니다. ("대충 써도 잘 맞는" 모델의 대명사)
- **단점**: SVM보다 훈련 속도가 느릴 수 있고, 모델이 완전히 블랙박스라 해석이 어렵습니다.

8 BBC 뉴스 분류 실습 사례 연구

강의에서 다룬 BBC 뉴스 분류 실습을 통해, 이 모델들이 불균형 데이터에서 어떻게 작동하는지 비교 분석합니다.

- **문제**: 2,225개의 BBC 뉴스 기사 분류
- **데이터**: 5개 카테고리 (스포츠, 비즈니스, 정치, 테크, 연예)
- **실습 목표**: "테크" 기사(401개)와 "비-테크" 기사(1824개)로 이진 분류 (약 1:4.5 불균형)
- **검증 방법**: 5-겹 교차 검증 (K-Fold CV)을 사용한 `GridSearchCV`
- **평가 지표**: 정확도(Accuracy), **재현율(Recall/Sensitivity)**, **특이도(Specificity)**

Table 4: *BBC* 뉴스 분류 모델별 성능 비교 (*5-Fold CV*)

모델 (Model)	최적 하이퍼파라미터	정확도(Acc.)	재현율(Recall)	특이도(Spec.)	**분석 및 평가**
Naive Bayes	(기본값)	89.6%	43% (낮음)	**100%**	**함정에 빠짐**. 정확도는 높아 보이나, 재현율이 43%라는 것은 **테크 기사의 절반 이상을 놓쳤다**는 의미. 특이도가 100%인 것은, 모델이 그냥 "비-테크"로만 예측했음을 시사.
KNN	`k=5` (기본값)	97%	94%	98%	매우 우수한 성능. 재현율과 특이도 모두 높음.
KNN (Optimized)	`k=19`	98%	94%	99%	`k=19` (더 넓은 이웃)가 기본값보다 약간 더 안정적인 성능을 보임.
Logistic Reg.	(기본값)	90.7%	69% (낮음)	**100%**	Naive Bayes와 유사. 재현율은 조금 낮지만 여전히 다수 클래스인 "비-테크"로 예측하는 경향이 매우 강함.
SVM (Default)	'kernel='rbf', 'C=1'	92.5%	70%	**100%**	Logistic Regression과 거의 동일한 문제 발생.
SVM (Optimized)	`C=10, kernel='linear'`	**98.8%**	**97%** **(Best)**	**99%**	**압도적인 1위.** 높은 `C` 값과 `linear` 커널이 이 고차원 TF-IDF 데이터에 가장 적합했음. 테크 기사를 **거의 놓치지 않으면서(재현율 97%)** 비-테크 기사도 완벽하게(특이도 99%) 분리해냄.
Random Forest	(기본값)	98.4%	89% (Slightly low)	**100%**	매우 좋은 성능. 하지만 SVM(Optimized)에 비해 재현율이 다소 낮아(89%), 일부 테크 기사를 놓침.

□ 핵심 요약

실습 결론

- **평가 지표의 중요성**: 단순 정확도만 봤다면 90%대의 Naive Bayes나 Logistic Regression도 좋아 보였겠지만, **재현율(Recall)**을 확인해보니 이 모델들은 "테크" 기사를 거의 찾지 못하는 쓸모없는 모델이었음을 알 수 있었습니다.
- **하이퍼파라미터 튜닝의 중요성**: SVM은 **기본값(rbf, C=1)**으로는 엉뚱한 결과(재현율 70%)를 냈지만, `GridSearchCV`를 통해 **최적의 파라미터(linear, C=10)**를 찾자 모든 모델 중 가장 완벽한 성능(재현율 97%)을 보여주었습니다.
- **고차원 텍스트와 Linear SVM**: TF-IDF로 변환된 텍스트 데이터는 수만 차원의 **초고차원(High-dimensional)** 데이터입니다. 이런 초고차원 공간에서는 데이터가 **선형(linear)으로 분리 가능**할 확률이 매우 높습니다. 이것이 복잡한 'rbf' 커널보다 단순한 **'linear' 커널 SVM**이 이 문제에서 최고의 성능을 낸 이유입니다.

9 학습 체크리스트 및 FAQ

9.1 학습 체크리스트

모델을 만들기 전, 아래 항목들을 점검해보세요.

텍스트 데이터를 숫자로 변환했는가? (e.g., `TfidfVectorizer`)

훈련 데이터와 테스트 데이터를 분리했는가?

데이터셋이 크지 않다면 (e.g., 10만 건 이하), K-Fold CV 사용을 고려했는가?

K-Fold CV 사용 시, `Pipeline`을 구성하여 데이터 누수(Leakage)를 방지했는가?

분류하려는 클래스 간의 비율이 불균형한가? (e.g., 9:1, 8:2)

(불균형하다면) '정확도(Accuracy)' 대신 '정밀도(Precision)'와 '재현율(Recall)'을 확인했는가?

KNN, SVM, Random Forest 같은 모델의 하이퍼파라미터를 튜닝했는가? (e.g., `GridSearchCV`)

9.2 FAQ (자주 묻는 질문)

Q: Naive Bayes와 Logistic Regression이 왜 이 문제에서 실패했나요?

A: 데이터가 "비-테크"(1824개) 쪽으로 심하게 **불균형**했기 때문입니다. 이 모델들은 손실(loss)을 최소화하는 과정에서, "일단 다수 클래스인 '비-테크'라고 예측하면 82%는 맞힌다"는 **쉬운 길(local minima)**에 빠지기 쉽습니다. 특이도(Specificity)가 100%라는 것은, 이 모델들이 "테크" 기사를 거의 무시하고 "비-테크"로만 예측했음을 보여줍니다.

Q: KNN에서 `k=19`가 `k=5`보다 좋은 이유는 무엇인가요?

A: `k=5`는 너무 **지역적(local)**인 정보만 봅니다. 만약 내 주변 5명만 보고 투표한다면, 그 5명이 우연히 이상한 의견을 가졌을 때(노이즈) 나의 예측도 흔들리기 쉽습니다. `k=19`는 더 **넓은(global)** 동네의 의견을 반영합니다. (e.g., 19명 중 2 3명이 이상해도 다수결에 큰 영향 없음) 이 데이터에서는 `k=19` 정도가 노이즈의 영향을 받지 않고 데이터의 전반적인 패턴을 더 잘 반영하는 "최적의 동네 크기"였던 것입니다.

Q: SVM에서 왜 `linear` 커널이 `rbf` (비선형) 커널보다 좋았나요?

A: **차원의 저주(Curse of Dimensionality)의 역설**입니다. TF-IDF 벡터는 **수만 차원**의 초고차원 공간에 존재합니다. 저차원(2D, 3D)에서는 데이터가 복잡하게 얽혀있어 선으로 나누기 어렵지만, **초고차원 공간**으로 가면 데이터 포인트 사이의 거리가 매우 멀어져서, 마치 텅 빈 우주에 점들이 흩뿌려진 것처럼 됩니다. 이런 초고차원 공간에서는 **단순한 선(초평면) 하나**만으로도 두 클래스를 분리할 수 있게 될 확률이 **매우 높아집니다.**
따라서 굳이 `rbf` 같은 복잡한 비선형 커널로 경계선을 꼬아줄 필요 없이, 가장 단순하고 빠른 **`linear` 커널**이 오히려 더 좋은 성능을 낸 것입니다.

Q: 최종 모델은 5개 모델의 평균인가요?

A: **아닙니다.** K-Fold CV는 **평가 및 하이퍼파라미터 탐색**을 위한 과정입니다. 일단 `GridSearchCV`를 통해 "SVM + C=10 + kernel='linear' 조합이 1등이다"라는 사실을 알아냈다면, K-Fold CV

는 거기서 임무가 끝납니다.
우리가 고객에게 배포할 **최종 모델**은, 이 1등 조합(SVM, C=10, `linear`)을 설정으로 하여 **전체 훈련 데이터(Train+Validation)를 모두 사용해 단 한 번** 훈련시킨, **단 하나의 모델**입니다.

10 빠른 훑어보기 (1-Page Quick Review)

□ 핵심 요약

텍스트 분류의 3단계 핵심 프로세스

1. **Vectorize (벡터화)**: 텍스트를 숫자로 변환 (`TfidfVectorizer`)
2. **Validate (검증)**: 모델의 "진짜 실력"을 측정. 데이터가 적으면 **K-Fold CV** 사용. 이때 **Pipeline**으로 데이터 누수 방지는 필수.
3. **Evaluate (평가)**: 모델의 성적표 확인.

불균형 데이터 평가의 제1원칙

클래스 비율이 9:1처럼 불균형하면, **절대 '정확도(Accuracy)'를 믿지 마세요.** "멍청한" 모델도 90%의 정확도를 달성할 수 있습니다.

반드시 **혼동 행렬(Confusion Matrix)**을 열고, 목적에 따라 **정밀도(Precision)**와 **재현율(Recall)**을 확인해야 합니다.

□ 고전적 모델 치트 시트 (Model Cheat Sheet)

모델	핵심 아이디어	언제 사용하는가?
Naive Bayes	"모든 단어는 독립"이라는 순진한 가정. ($P(A \cap B) = P(A) \times P(B)$)	빠르고 간단한 **베이스라인**이 필요할 때. (e.g., 스팸 필터)
KNN	"가장 가까운 k명의 이웃에게 물어보고 **다수결**로 결정."	모델이 직관적이어야 할 때. (e.g., 추천 시스템) **정규화 필수!**
Logistic Reg.	뉴런 1개. 선형 결합(z)을 시그모이드 함수로 압축해 **0 1 확률** 출력.	분류 결과가 "확률"로 나와야 할 때. 선형 경계가 잘 먹힐 때.
SVM	두 클래스 사이의 **"도로 폭(Margin)"을 최대화**하는 선을 찾음. + **커널 트릭** (e.g., `'rbf'`)	**텍스트(TF-IDF) 분류** 같이 고차원 데이터에 매우 강력함. 비선형 데이터도 차원 트릭으로 해결 가능. (범용성 높음)
Random Forest	과적합된 "전문가(트리)" 수백 명을 **무작위**로 만들어 **투표**시킴.	**과적합을 피하는** 가장 안정적인 방법. (성능이 웬만해선 잘 나옴)