Ch.11 Machine Learning

Model

- 서로 다른 변수들 사이에 존재하는 수학적(또는 확률적) 관계의 특정을 의미합니다.

Machine Learning

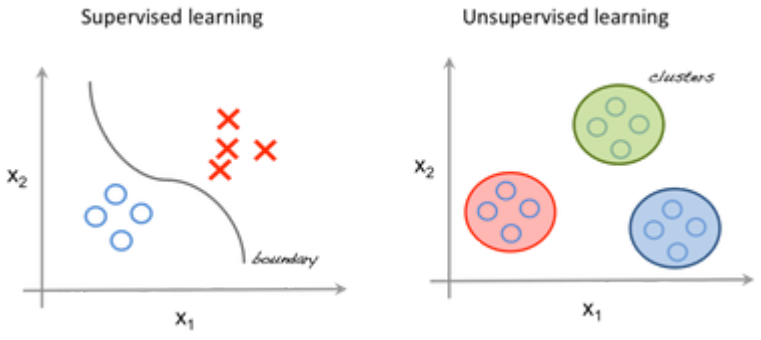
- 데이터에서 학습한 모델을 만들고 사용합니다.
- 예측 모델링 또는 데이터 마이닝이라고도 합니다
- Example
  - 전자 메일 메시지가 스팸인지 여부 예측
  - 신용카드 거래 사기 여부 예측
  - 쇼핑객이 어떤 광고를 클릭할 가능성이 가장 높은지 예측하기
  - 어떤 축구팀이 슈퍼볼에서 우승할지 예측하기

Supervised Model (지도학습 모델)

- 학습할 수 있는 정답이 표시된 데이터 세트가 있습니다
  - 회귀(Regression)
  - 분류(Classification)

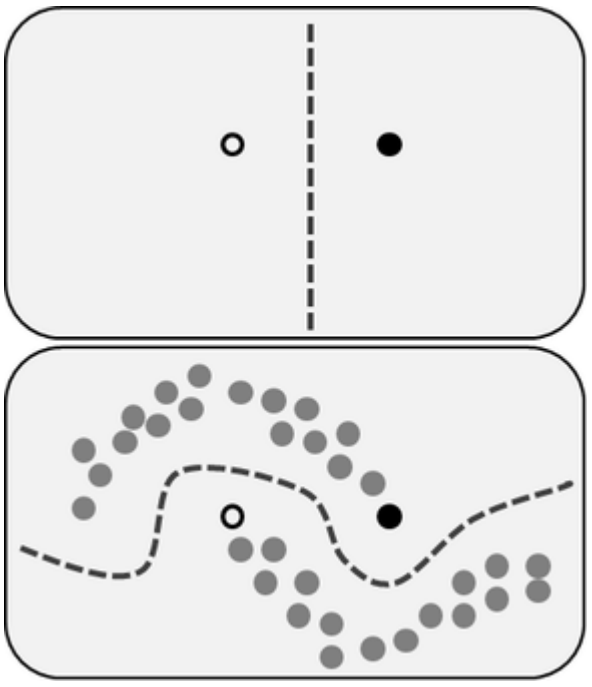
Unsupervised Model (비지도학습 모델)

- 이러한 label(레이블)이 없습니다
  - Clustering (클러스터링)
  - Anomaly detection (이상 탐지)
  - Association
  - Topic Modelling (토픽 모델링)
  - Autoencoders (오토인코더)



Semisupervised Learning (준지도 학습)

- 소량의 직접 지시(예: 아동기 대상에 대한 부모의 라벨링)와 많은 양의 라벨링이 결합된 경험
- 일부 데이터에만 레이블이 지정됩니다
  - 예: CT 스캔 또는 MRI.
  - 훈련된 방사선사는 종양 또는 질병에 대한 스캔의 작은 부분 집합에 레이블을 지정할 수 있습니다.
  - 기계 학습은 유사한 스캔을 클러스터링하고 레이블을 지정합니다



```
from sklearn.datasets import load_iris
import pandas as pd

iris = load_iris()
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_df.head(5)

# iris_df.head(5)를 사용하여 데이터프레임의 첫 5개 행을 출력.
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
iris_target_df = pd.DataFrame(iris.target)
iris_target_df.head(2)
```

	0
0	0
1	0

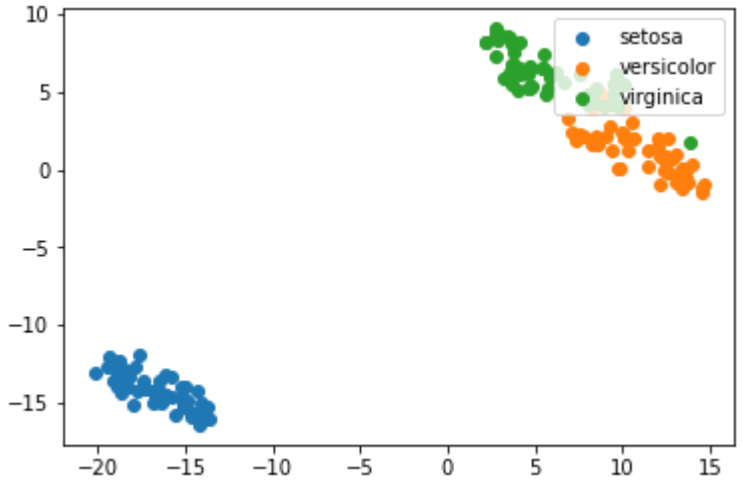
```
iris.target_names

# array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
import numpy as np

def plot_tsne(data, target, target_names, learning_rate=100, perplexity=30):
    # t-SNE 모델 생성 및 데이터 적합
    model = TSNE(learning_rate=learning_rate, perplexity=perplexity)
    embedded = model.fit_transform(data)

    # 2차원 좌표 추출
```

```
    xs = embedded[:, 0]
    ys = embedded[:, 1]

    # 각 타겟 레이블에 대해 산점도 그리기
    for t in np.unique(target):
        i = np.where(target == t)
        plt.scatter(xs[i], ys[i], label=target_names[t])

    # 레전드 추가
    plt.legend(loc=1)

    # 플롯 표시
    plt.show()
```

- t-SNE를 사용하여 데이터를 2차원으로 시각화합니다.
- Parameters:
- data (array-like): 입력 데이터.
- target (array-like): 각 데이터 포인트의 타겟 레이블.
- target_names (list): 타겟 레이블의 이름들.
- learning_rate (float): t-SNE 학습률.
- perplexity (float): t-SNE 퍼플렉서티.

```
plot_tsne(iris.data, iris.target, iris.target_names)
# print(iris.data)
```



YELP 데이터 세트

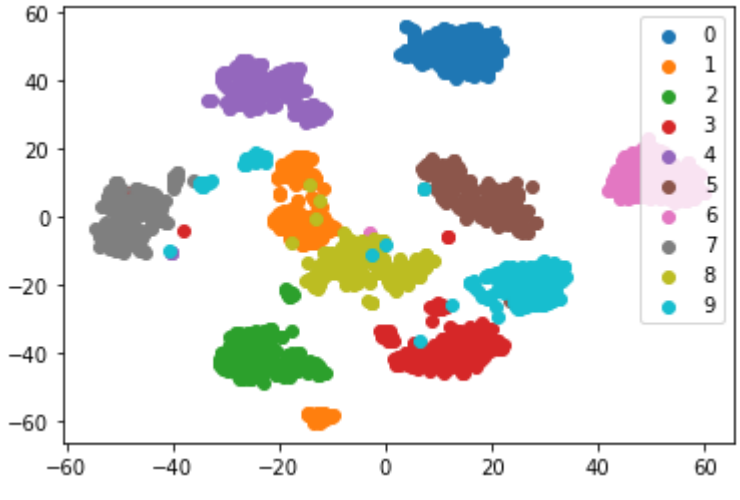
- 훈련에 iris.data를 사용하면 비지도 학습입니다
- iris.data와 iris.target을 사용하면 지도학습이 됩니다
- 지도 학습을 위한 데이터 세트를 준비하는 것은 매우 비용이 많이 듭니다.
- YELP 데이터 세트도 생각해 보세요.

  https://www.yelp.com/dataset/challenge

Manifold Learning or PCA is a type of unsupervised laerning

```
from sklearn import datasets
```

```
digits = datasets.load_digits()
plot_tsne(digits.data, digits.target, digits.target_names)
```



What we usually do (우리가 주로 하는 일)

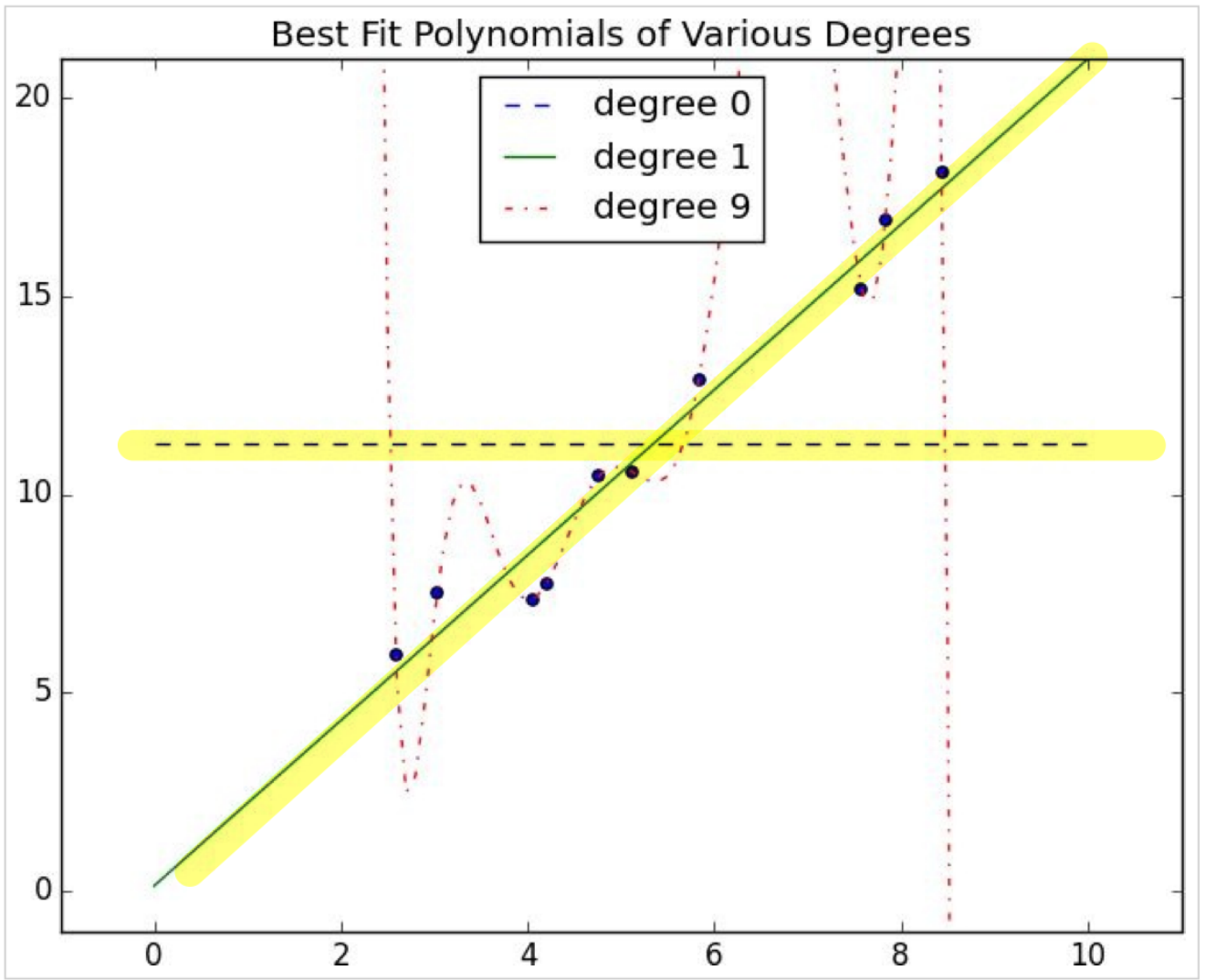
- *모수화된* 모형군을 선택한 다음 데이터를 사용하여 최적의 모수를 학습합니다.
- 예:
  - 사람의 키는 (대략) 몸무게의 선형 함수이고 그 선형 함수가 무엇인지를 알기 위해 데이터를 사용합니다.
  - 우리는 의사결정 나무가 환자가 어떤 질병을 가지고 있는지 진단하고 데이터를 사용하여 그러한 나무를 "최적"으로 학습하는 좋은 방법이라고 가정할 수 있습니다

Overfitting (과적합)

- 기계 학습의 일반적인 위험은 Overfitting(과적합)입니다.
- 학습한 데이터에 대해 우수한 성능을 발휘하는 모델 제작
- 하지만 새로운 데이터에는 잘 Normalized(일반화)되지 않습니다.
- 여기에는 Learning Noise(**학습 노이즈**)가 데이터에 포함될 수 있습니다.
- 또는 원하는 출력에 대해 실제로 예측 가능한 요소가 아닌 특정 입력을 식별하는 학습이 포함될 수 있습니다.

Underfitting (언더피팅)

- Train 데이터에서도 잘 수행되지 않는 모델 제작
- 일반적으로 이런 일이 발생하면 모델이 충분하지 않다고 판단하고 더 나은 모델을 계속 찾고 있습니다.



Underfitting (과소적합)

- 수평선은 가장 적합도가 높은 0(즉, 상수) 다항식을 나타냅니다. 이 다항식은 훈련 데이터에 매우 적합하지 않습니다.

Overfitting (과적합)

- 최적 적합도 9(즉, 10-모수) 다항식은 모든 훈련 데이터 포인트를 정확하게 통과하지만, 데이터 포인트를 몇 개 더 선택하면 데이터 포인트를 많이 놓칠 수 있습니다.

Best fitting (가장 잘 맞음)

- 그리고 1도 선은 균형이 잘 맞습니다. 모든 점에 상당히 가깝고, (이러한 데이터가 대표적인 경우) 그 선은 새로운 데이터 점에도 가까울 것입니다.

Model complexity(모델 복잡도)

- 분명히 너무 복잡한 모델은 Overfitting(과적합)으로 이어집니다
- 그리고 그들이 훈련받은 데이터 이상으로 Generalize(일반화)하지 마십시오.

Occam's razor (오캠 면도기)

- Occam이 말한 것과 정확히 일치합니다: "필요한 것보다 더 많은 것이 사용되어서는 안 됩니다."
- "복잡한 솔루션보다 더 간단한 솔루션이 정확할 가능성이 높습니다."

https://en.wikipedia.org/wiki/Occam's_razor#Science_and_the_scientific_method

To avoid overfitting(과적합을 피하기 위해)

- 다양한 데이터를 사용하여 모형을 훈련하고 모형을 검정합니다.
  - Training dataset
  - Test dataset
- 여러 모델 중에서 선택해야 할 경우:

- Training dataset
- Validation set for choosing among trained models (훈련된 모델 중에서 선택할 수 있는 유효성 검사 세트)
- Test dataset

```
import random

def split_data(data, prob):
    results = [], []
    for row in data:
        # 랜덤 값이 prob보다 작으면 첫 번째 리스트에 추가, 아니면 두 번째 리스트에 추가
        results[0 if random.random() < prob else 1].append(row)
    return results
```

- 데이터를 [prob, 1 - prob] 비율로 나눕니다.
- Parameters:
- data (list): 데이터를 담은 리스트.
- prob (float): 데이터를 나누는 비율.
- Returns: tuple: 나눠진 두 데이터 리스트.

```
def train_test_split(x, y, test_pct):
    data = list(zip(x, y))                          # 특성과 레이블을 쌍으로 묶음
    train, test = split_data(data, 1 - test_pct)  # 데이터셋을 훈련과 테스트 세트로 나눔
    x_train, y_train = list(zip(*train))          # 훈련 데이터 쌍을 분리
    x_test, y_test = list(zip(*test))             # 테스트 데이터 쌍을 분리
    return x_train, x_test, y_train, y_test
```

Parameters:

- x (list): 특성 데이터 리스트.
- y (list): 레이블 데이터 리스트.
- test_pct (float): 테스트 데이터 비율.
- Returns: tuple: 훈련 데이터(x_train, y_train)와 테스트 데이터(x_test, y_test) 리스트.

Use sklearn's train_test_split

```
import numpy as np
from sklearn.model_selection import train_test_split

# 특성 데이터 X와 레이블 데이터 y 생성
X, y = np.arange(12).reshape((6, 2)), list(range(6))

# 특성 데이터와 레이블 데이터 출력
print("X:")
print(X)
print("y:")
print(y)

# 데이터셋을 훈련 세트와 테스트 세트로 나누기 (테스트 비율 33%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)

# 훈련 세트와 테스트 세트 출력
```

```
print("X_train:")
print(X_train)
print("y_train:")
print(y_train)
print("X_test:")
print(X_test)
print("y_test:")
print(y_test)
```

```
[[ 0  1]
 [ 2  3]
 [ 4  5]
 [ 6  7]
 [ 8  9]
 [10 11]]
[0, 1, 2, 3, 4, 5]
[[10 11]
 [ 8  9]
 [ 0  1]
 [ 2  3]] [5, 4, 0, 1]
[[4 5]
 [6 7]] [2, 3]
```

Story (이야기) - Example

- 저는 신생아가 백혈병에 걸릴 가능성을 98% 이상의 정확도로 예측하는 값싸고 비침습적인 검사를 개발했습니다.
- 제 변호사는 그 테스트가 특허를 받을 수 없다고 저를 설득했습니다.
- 진실은 "백혈병은 아기의 이름이 루크인 경우에만 예측하라"는 것인데, 이는 "백혈병"처럼 들립니다
- 일반적으로 "accuracy"을 사용하여 모델이 얼마나 우수한지 측정하지 않는 이유입니다.

모형이 Binary(이진) 판정을 내린다고 가정합니다.

- 이메일 스팸인가요? → Positive, Negative
- 이 후보자를 고용해야 하나요?
- 이 항공 여행자는 비밀리에 테러리스트입니까?
- AI가 용의자의 유무를 판단합니다 (FP)
- 의사는 환자가 암에 걸렸는지 아닌지를 결정합니다 (FN)

스팸 필터링 예제

- True positive: "이 메시지는 스팸이며, 우리는 스팸을 정확하게 예측했습니다."
- False positive : "이 메시지는 스팸이 아니라 스팸을 예측했습니다."
- False negative: "이 메시지는 스팸이지만 스팸이 아닐 것으로 예상했습니다."
- True negative: "이 메시지는 스팸이 아니며 스팸이 아닐 것으로 정확하게 예측했습니다."

Confusion Matrix:

	Spam	**not Spam**
predict “Spam”	True Positive	False Positive

predict “Not Spam”	False Negative	True Negative

- 요즘에는 약 1,000명의 아기들 중 5명의 아기들이 루크라고 불립니다.
- 그리고 백혈병의 평생 유병률은 약 1.4%, 즉 1,000명당 14명입니다.

	leukemia	**no leukemia**	**total**
“Luke”	70	4,930	5,000
not “Luke”	13,930	981,070	995,000
total	14,000	986,000	1,000,000

Correctness

정확성

- 정확한 예측의 비율:

```
def accuracy(tp, fp, fn, tn):
    """
    Parameters:
    tp (int): True Positives (참 긍정)
    fp (int): False Positives (거짓 긍정)
    fn (int): False Negatives (거짓 부정)
    tn (int): True Negatives (참 부정)

    Returns: float: 정확도
    """
    correct = tp + tn   # 올바르게 분류된 샘플 수
    total = tp + fp + fn + tn  # 전체 샘플 수
    return correct / total  # 정확도 계산

# 정확도 계산 및 출력
print("accuracy(70, 4930, 13930, 981070) = {}".format(accuracy(70, 4930, 13930, 981070)))
# 예상 결과: accuracy(70, 4930, 13930, 981070) = 0.98114
```

Precision(정밀도), Recall(리콜), f1_score

- 정확도는 우리의 긍정적인 예측이 얼마나 정확했는지를 측정합니다
- 리콜은 모델이 식별한 긍정적인 부분을 측정합니다
- F1 점수: 정밀도와 리콜이 결합되었습니다
    - 이것은 정밀도와 리콜의 조화 평균이며 반드시 그들 사이에 있습니다

```
def precision(tp, fp, fn, tn):
    return tp / (tp + fp)

def recall(tp, fp, fn, tn):
    return tp / (tp + fn)

def f1_score(tp, fp, fn, tn):
    p = precision(tp, fp, fn, tn)
    r = recall(tp, fp, fn, tn)
    return 2 * p * r / (p + r)
```

- `return tp / (tp + fp)`는 정밀도를 계산하여 반환합니다.
- `return tp / (tp + fn)`는 재현율을 계산하여 반환합니다.
- `return 2 * p * r / (p + r)`는 F1 점수를 계산하여 반환합니다.

```
print("precision(70, 4930, 13930, 981070) = {}".format(precision(70, 4930, 13930, 981070)))
print("recall(70, 4930, 13930, 981070) = {}".format(recall(70, 4930, 13930, 981070)))
print("f1_score(70, 4930, 13930, 981070) = {}".format(f1_score(70, 4930, 13930, 981070)))
```

```
precision(70, 4930, 13930, 981070) = 0.014
recall(70, 4930, 13930, 981070) = 0.005
f1_score(70, 4930, 13930, 981070) = 0.00736842105263158
```

$F\beta$ score

```
# 베타가 2일 때, 재현율(Recall)이 정밀도(Precision)보다 두 배 더 중요합니다.

# 정밀도와 재현율 값을 설정
p = 0.5
r = 0.5

# F1 점수를 계산
f1 = 2 * p * r / (p + r)
f1

# 0.5
```

```
# 베타가 2일 때, 재현율(Recall)이 정밀도(Precision)보다 두 배 더 중요합니다.

# 정밀도와 재현율 값 설정
p = 0.5 * 0.6  # 40% 감소
r = 0.5 * 1.2  # 20% 증가

# 베타가 2일 때의 F2 점수 계산
beta = 2
f2 = (1 + beta**2) * p * r / (beta**2 * p + r)
f2

# 0.5
```

Example: Tragedy of an island

```
population in an island: 300,000
bad guys: 300
# of total dead including all bad guys: 30,000
```

```
# 베타가 2일 때, 재현율(Recall)이 정밀도(Precision)보다 두 배 더 중요합니다.

# 정밀도와 재현율 값 설정
p = 0.5 * 0.6  # 40% 감소
r = 0.5 * 1.2  # 20% 증가

# 베타가 2일 때의 F2 점수 계산
beta = 2
f2 = (1 + beta**2) * p * r / (beta**2 * p + r)
f2
```

```
# 0.3181818181818181
```

```
# True Positives, False Positives, False Negatives, True Negatives 값 설정
tp = 300
fp = 30000 - 300
fn = 0
tn = 300000 - 30000

# 정밀도, 재현율, F1 점수 계산 및 출력
print(f"precision({tp}, {fp}, {fn}, {tn}) = {precision(tp, fp, fn, tn):.5f}")
print(f"recall({tp}, {fp}, {fn}, {tn}) = {recall(tp, fp, fn, tn):.5f}")
print(f"f1_score({tp}, {fp}, {fn}, {tn}) = {f1_score(tp, fp, fn, tn):.5f}")
```

```
precision(300, 29700, 0, 270000) = 0.01
recall(300, 29700, 0, 270000) = 1.0
f1_score(300, 29700, 0, 270000) = 0.019801980198019802
```

💡 설명.

- **True Positives (TP)**: 실제로 악당이고, 모델이 악당으로 올바르게 분류한 수.
- **False Positives (FP)**: 실제로는 악당이 아니지만, 모델이 악당으로 잘못 분류한 수.
- **False Negatives (FN)**: 실제로 악당이지만, 모델이 악당이 아니라고 잘못 분류한 수.
- **True Negatives (TN)**: 실제로 악당이 아니고, 모델이 악당이 아니라고 올바르게 분류한 수.

- **전체 인구**: 300,000
- **악당 수 (bad guys)**: 300
- **총 사망자 수**: 30,000 (모든 악당 포함)

- **True Positives (TP)**:
  - 모든 악당이 사망한 것으로 가정하므로, TP = 300
- **False Positives (FP)**:
  - 사망자 중 악당이 아닌 사람의 수를 계산
  - 총 사망자 수 - TP = 30,000 - 300 = 29,700
  - 따라서 FP = 29,700
- **False Negatives (FN)**:
  - 악당 중에서 생존한 사람의 수를 계산
  - 모든 악당이 사망했으므로 FN = 0
- **True Negatives (TN)**:
  - 전체 인구에서 사망하지 않은 사람의 수를 계산
  - 전체 인구 - 총 사망자 수 = 300,000 - 30,000 = 270,000
  - 생존한 사람 중 악당이 아닌 사람의 수를 계산
  - TN = 270,000

```
def fbeta_score(tp, fp, fn, tn, beta=1):
    p = precision(tp, fp, fn, tn) # 정밀도 계산
    r = recall(tp, fp, fn, tn) # 재현율 계산
    return (1 + beta**2) * p * r / (beta**2*p + r)  # F-Beta 점수 계산
```

```
# 베타 값을 0부터 100까지 201개의 점으로 생성
betas = np.linspace(0, 100, 201)

# 각 베타 값에 대해 F-점수를 계산하고 출력
for beta in betas:
    print(f"fbeta_score({tp}, {fp}, {fn}, {tn}, {beta}) = {fbeta_score(tp, fp, fn, tn, beta)}")
```

- `for beta in betas` 루프를 사용하여 각 베타 값에 대해 F-점수를 계산합니다.
- `print(f"fbeta_score({tp}, {fp}, {fn}, {tn}, {beta}) = {fbeta_score(tp, fp, fn, tn, beta)}")`를 사용하여 각 베타 값에 대한 F-점수를 출력합니다.

---

sklearn compusion matrix 및 classification_report를 사용:

- sklearn 혼동은 "Actual(실제) 대 Predict(예측)" (sklearn)이라는 점에 유의하십시오
- Actual(실제) 대 Predict(예측)
- 교재 형식으로 읽고 싶다면 Confusion matrix(혼동 행렬)을 Tranpose(전치)합니다.

```
from sklearn.metrics import confusion_matrix, classification_report

# 실제 레이블과 예측 레이블 설정
y_true = ["cat", "ant", "cat", "cat", "ant", "cat"]
y_pred = ["ant", "ant", "cat", "cat", "ant", "cat"]

# 혼동 행렬 계산 및 전치
conf_matrix = confusion_matrix(y_true, y_pred, labels=["ant", "cat"]).T

# 혼동 행렬 출력
print(conf_matrix)

# Result
# [[2 1]
#  [0 3]]
```

	"ant"	**"cat"**
predict “ant”	2	1
predict “cat”	0	3

- 개미를 예측하는 데 문제가 있는 경우 Confusion Matrix (혼동 행렬)은 다음과 같습니다:

	"ant"	**"not ant"**
predict “ant”	2 (TP)	1 (FP)
predict “not ant”	0 (FN)	3 (TN)

- cat을 예측하는 문제라면 혼동 행렬은 다음과 같습니다:

	"not cat	**"cat"**
predict “not cat”	2 (TN)	1 (FP)
predict “cat”	0 (FP)	3 (TN)

```
print(classification_report(y_true, y_pred, labels=["ant", "cat"]))
```

```
              precision    recall  f1-score   support

         ant       0.67      1.00      0.80         2
         cat       1.00      0.75      0.86         4

    accuracy                           0.83         6
   macro avg       0.83      0.88      0.83         6
weighted avg       0.89      0.83      0.84         6
```

ant

- **precision (정밀도)**: 0.67
  - 모델이 "ant"로 예측한 것들 중 실제로 "ant"인 것들의 비율.
  - 계산: $\frac{TP}{TP+FP} = \frac{2}{2+1} = 0.67$
- **recall (재현율)**: 1.00
  - 실제 "ant"인 것들 중 모델이 "ant"로 올바르게 예측한 것들의 비율.
  - 계산: $\frac{TP}{TP+FN} = \frac{2}{2+0} = 1.00$
- **f1-score**: 0.80
  - 정밀도와 재현율의 조화 평균.
  - 계산: $2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = 2 \times \frac{0.67 \times 1.00}{0.67 + 1.00} = 0.80$
- **support**: 2
  - 실제 "ant"의 개수.

cat

- **precision (정밀도)**: 1.00
  - 모델이 "cat"으로 예측한 것들 중 실제로 "cat"인 것들의 비율.
  - 계산: $\frac{TP}{TP+FP} = \frac{3}{3+0} = 1.00$
- **recall (재현율)**: 0.75
  - 실제 "cat"인 것들 중 모델이 "cat"으로 올바르게 예측한 것들의 비율.
  - 계산: $\frac{TP}{TP+FN} = \frac{3}{3+1} = 0.75$
- **f1-score**: 0.86
  - 정밀도와 재현율의 조화 평균.
  - 계산: $2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = 2 \times \frac{1.00 \times 0.75}{1.00 + 0.75} = 0.86$
- **support**: 4
  - 실제 "cat"의 개수.

```
from sklearn.metrics import confusion_matrix, classification_report

y_true = ["cat", "ant", "cat", "cat", "ant", "cat", "bird", "bird"]
y_pred = ["ant", "ant", "cat", "cat", "ant", "cat", "bird", "bird"]
print(confusion_matrix(y_true, y_pred, labels=["ant", "cat", "bird"]).T)
print(classification_report(y_true, y_pred, labels=["ant", "cat", "bird"]))
```

```
[[2 1 0]
 [0 3 0]
 [0 0 2]]
              precision    recall  f1-score   support

         ant       0.67      1.00      0.80         2
         cat       1.00      0.75      0.86         4
        bird       1.00      1.00      1.00         2

    accuracy                           0.88         8
   macro avg       0.89      0.92      0.89         8
weighted avg       0.92      0.88      0.88         8
```

ant

- **precision**: 0.67
  - "ant"로 예측한 것 중 실제 "ant"인 것들의 비율.
- **recall**: 1.00
  - 실제 "ant"인 것 중 모델이 "ant"로 올바르게 예측한 비율.
- **f1-score**: 0.80
  - 정밀도와 재현율의 조화 평균.
- **support**: 2
  - 실제 "ant"의 개수.

cat

- **precision**: 1.00
  - "cat"로 예측한 것 중 실제 "cat"인 것들의 비율.
- **recall**: 0.75
  - 실제 "cat"인 것 중 모델이 "cat"로 올바르게 예측한 비율.
- **f1-score**: 0.86
  - 정밀도와 재현율의 조화 평균.
- **support**: 4
  - 실제 "cat"의 개수.

bird

- **precision**: 1.00
  - "bird"로 예측한 것 중 실제 "bird"인 것들의 비율.
- **recall**: 1.00
  - 실제 "bird"인 것 중 모델이 "bird"로 올바르게 예측한 비율.
- **f1-score**: 1.00
  - 정밀도와 재현율의 조화 평균.
- **support**: 2
  - 실제 "bird"의 개수.

- **accuracy**: 0.88
  - 전체 예측 중에서 맞춘 비율.
- **macro avg**: 0.89 (정밀도), 0.92 (재현율), 0.89 (f1-score)
  - 모든 클래스의 평균을 계산.
- **weighted avg**: 0.92 (정밀도), 0.88 (재현율), 0.88 (f1-score)
  - 각 클래스의 샘플 수를 가중치로 고려한 평균을 계산.

Precision(정밀도)와 Recall(리콜) 간의 균형

- 조금이라도 자신이 있을 때 "Yes"를 예측하는 모델은 아마도 리콜은 높지만 정밀도는 낮을 것입니다;
- 극도로 자신이 있을 때만 "Yes"를 예측하는 모델은 리콜이 낮고 정밀도가 높을 가능성이 있습니다.
- 또는 이것을 False 및 True 사이의 trade-off로 생각할 수 있습니다.
  - "Yes"라고 너무 자주 말하는 것은 당신에게 많은 False Positive을 줄 것입니다
  - 너무 자주 "No"라고 말하는 것은 당신에게 많은 False Negative을 줄 것입니다

Cost matrix

- Cost matrix는 confusion matrix와 유사하지만, 여기서는 잘못된 예측이나 올바른 예측에 대한 비용을 계산합니다.

	Predator	not Predator
predict “Predator”	-1	1
predict “Not Predator”	100	0

- 인간이 100명의 Predator와 1,000명의 non-predator를 만나는 세상에 살고 있다고 가정해 봅시다.

Model 1:

	Predator	not Predator
predict “Predator”	99	999
predict “Not Predator”	1	1

- Precision = 9%
- Recall = 99%
- Model 1의 총 비용 = `99*(-1) + 999*(1) + 1*(100) + 1*(0) = 1000`

Model 2

	Predator	not Predator
predict “Predator”	1	1
predict “Not Predator”	99	999

- Precision = 50%
- Recall = 1%
- Model 2의 총 비용 = `1*(-1) + 1*(1) + 99*(100) + 999*(0) = 9900`
- Model 2는 진화 과정에서 생존할 확률이 낮습니다.
- 우리는 모두 Model 1의 인간입니다.
- 오늘날의 환경에서는 지나치게 민감한 모델입니다. Predator가 없습니다!!!

오늘날의 세상

- 우리는 인간이 2명의 Predator와 10,000명의 non-predator를 만나는 세상에 살고 있습니다.

Model 1:

	Predator	not Predator
predict "Predator"	2	9999
predict "Not Predator"	0	1

- Precision = 0.02%
- Recall = 100%
- Model 1의 총 비용 = `2*(-1) + 9999*(1) + 0*(100) + 1*(0) = 9997`

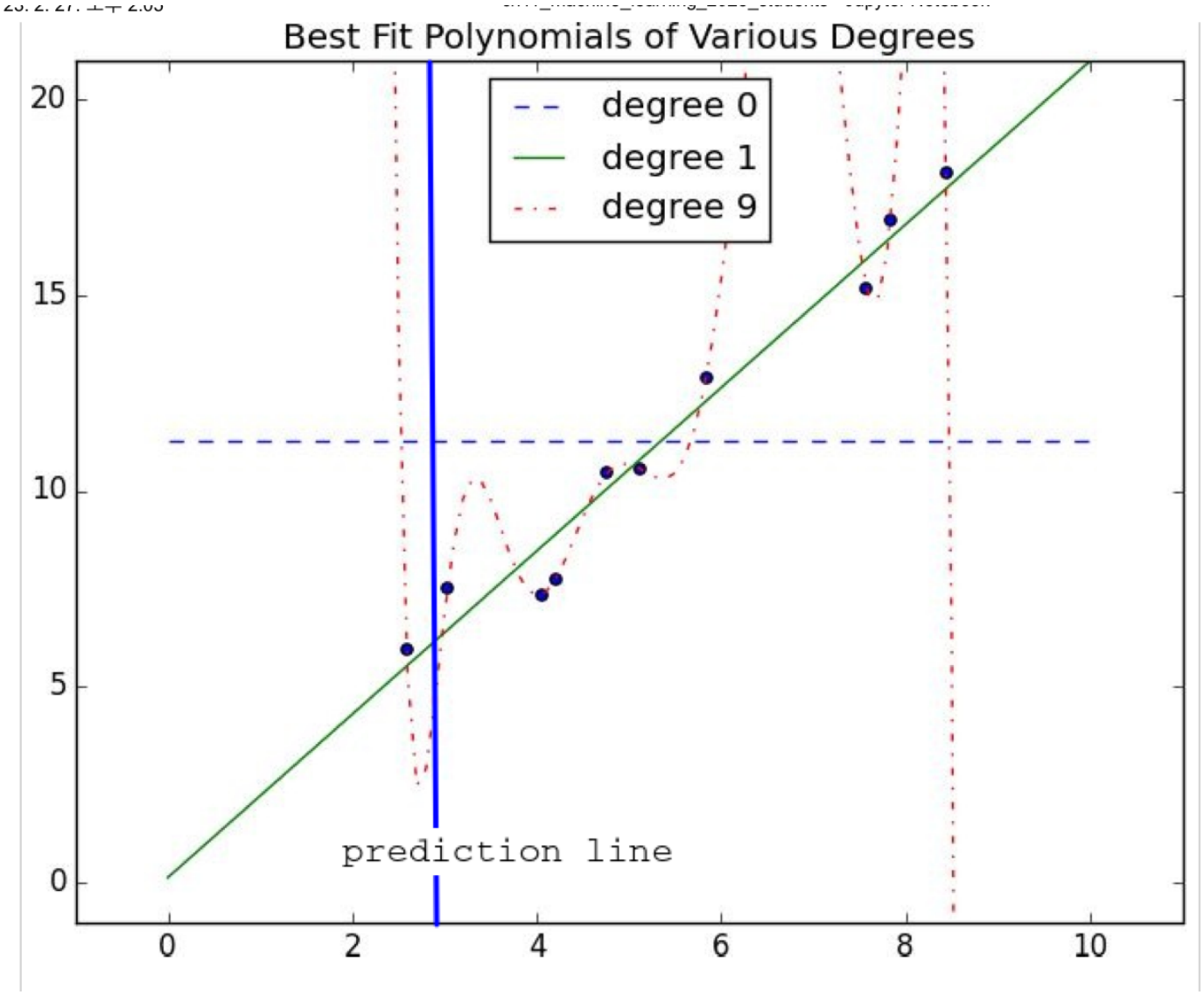
Model 2

	Predator	not Predator
predict "Predator"	1	1
predict "Not Predator"	1	9999

- Precision = 50%
- Recall = 50%
- Model 2의 총 비용 = `1*(-1) + 1*(1) + 1*(100) + 9999*(0) = 100`
- Model 1은 진화 과정에서 생존할 가능성이 더 낮습니다.

Bias-Variance Trade-off

- 두 가지 모두 **여러 번 모델을 재학습시킬 때** (같은 큰 집단에서 추출된 다른 학습 데이터 세트를 사용하여) 발생하는 결과를 측정합니다.
- Bias(편향)는 모델의 평균 예측값과 우리가 예측하려는 실제 값 간의 차이를 의미합니다.
- Variance(분산)는 주어진 데이터 포인트에 대한 모델 예측의 변동성을 의미합니다(데이터의 분포를 나타내는 값).



- 예측 선(prediction line)의 편향은 무엇인가요? → 모델의 평균 예측값과 실제 값 간의 차이를 의미
- 예측 선(prediction line)의 분산은 무엇인가요? → 주어진 데이터 포인트에 대한 모델 예측의 변동성을 의미

💡 결론

- **Degree 0 모델**: 높은 편향, 낮은 분산 → 과소적합(underfitting)
- **Degree 1 모델**: 중간 정도의 편향 및 분산 → 적절한 모델 (데이터에 따라 다름)
- **Degree 9 모델**: 낮은 편향, 높은 분산 → 과적합(overfitting)

편향(Bias)과 분산(Variance)의 예시

Degree 0 모델:

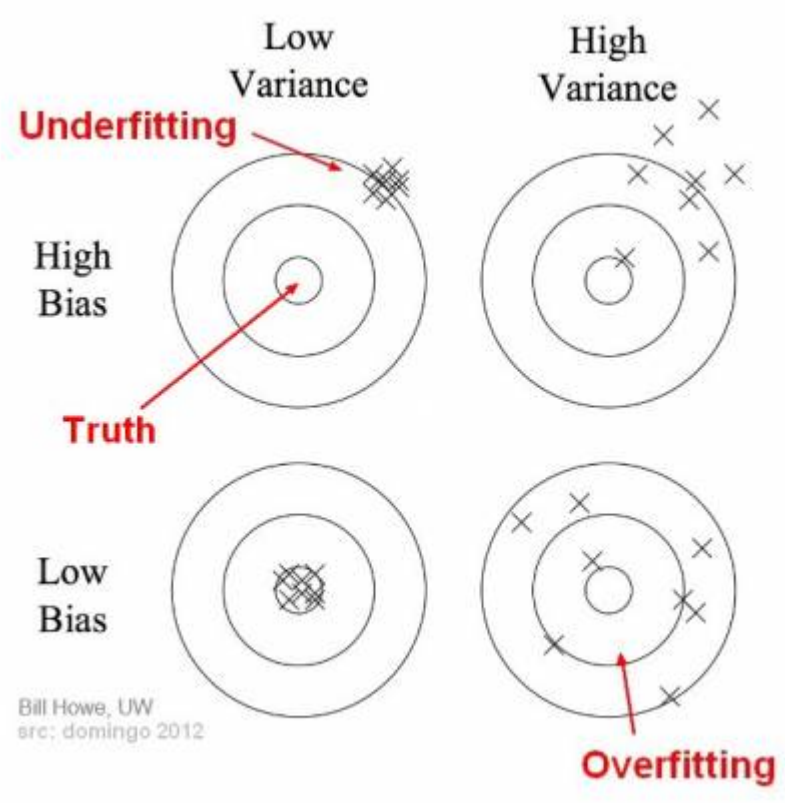
- Degree 0 모델은 거의 모든 학습 세트에 대해 많은 실수를 저지르므로, **높은 편향**을 가지고 있습니다.
- 하지만, 두 개의 무작위로 선택된 학습 세트는 매우 유사한 평균값을 가지므로, 모델도 유사하게 나올 것입니다. 따라서 **낮은 분산**을 가집니다.
- **높은 편향과 낮은 분산은 일반적으로 과소적합(underfitting)을 나타냅니다.**

Degree 9 모델:

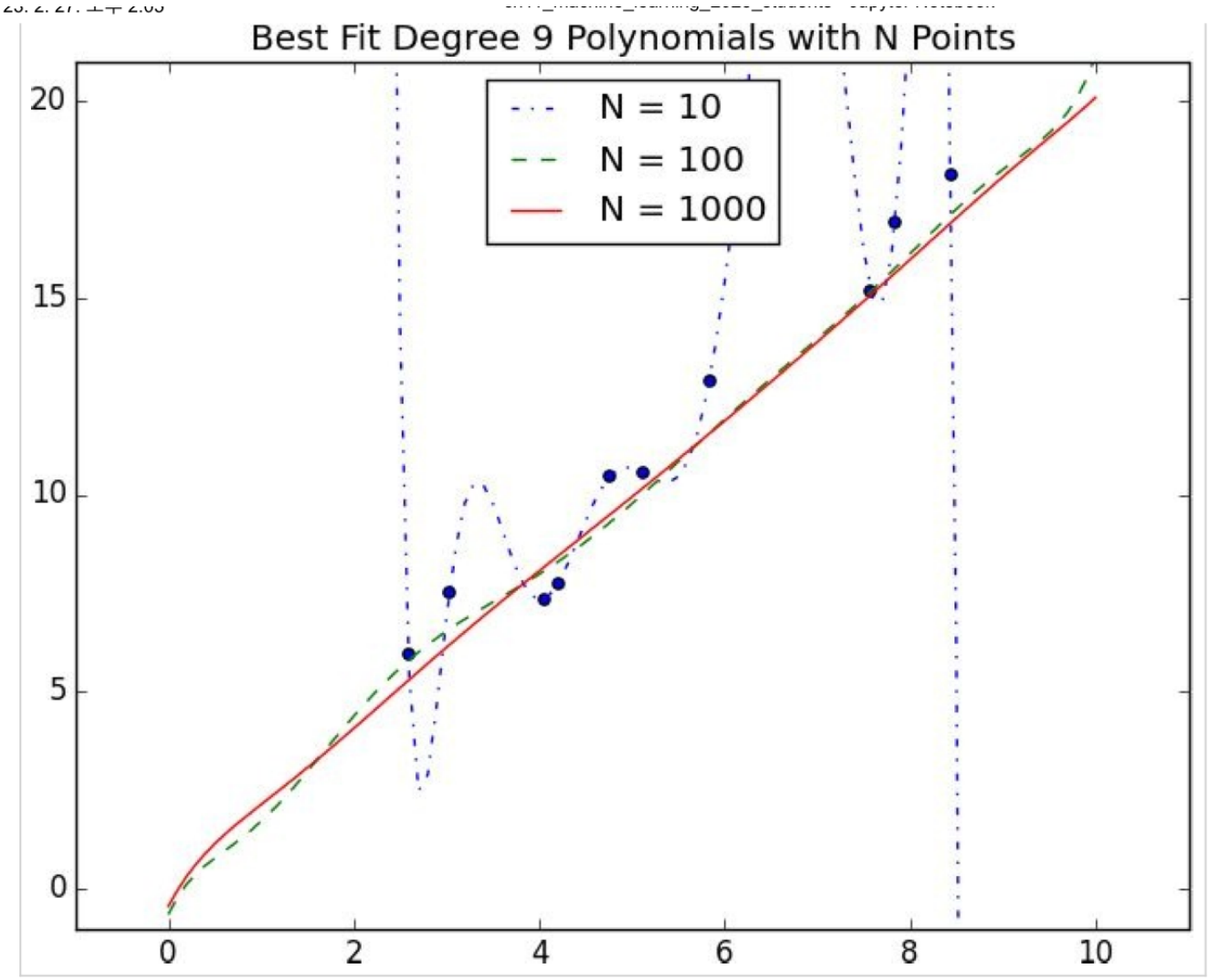
- Degree 9 모델은 학습 세트를 완벽하게 맞춥니다.
- **편향이 매우 낮지만, 분산이 매우 높습니다** (두 개의 학습 세트가 매우 다른 모델을 만들어낼 가능성이 높기 때문입니다). **이는 과적합(overfitting)을 나타냅니다.**

해결 방법

- 모델이 높은 편향을 가지고 있다면(즉, 학습 데이터에서도 성능이 좋지 않다면), 더 많은 특징을 추가해 볼 수 있습니다. Degree 0 모델에서 Degree 1 모델로 가는 것은 큰 개선을 보여줍니다.
- 모델이 높은 분산을 가지고 있다면, 특징을 제거해 볼 수 있습니다. 하지만 또 다른 해결책은 더 많은 데이터를 얻는 것입니다(가능하다면).



Degree 9 model with N points



데이터 크기와 모델 복잡성에 따른 적합(fitting)

- 우리는 degree 9 다항식을 서로 다른 크기의 샘플에 맞춥니다.
- 10개의 데이터 포인트를 기반으로 학습된 모델은 매우 불안정합니다(앞서 본 것처럼).
- 대신 100개의 데이터 포인트로 학습하면, 과적합이 훨씬 줄어듭니다.
- 1,000개의 데이터 포인트로 학습된 모델은 degree 1 모델과 매우 유사하게 보입니다.

데이터 양과 과적합

- 모델 복잡성을 일정하게 유지한 상태에서 데이터 양이 많을수록 과적합(overfitting)이 더 어려워집니다.
- 데이터가 많아지면 모델이 특정 데이터 포인트에 과적합되기 어려워지며, 더 일반화된 패턴을 학습하게 됩니다.

데이터 양과 편향

- 반면, 더 많은 데이터는 편향(bias) 문제를 해결하는 데 도움이 되지 않습니다.
- 모델이 데이터의 규칙성을 포착하기에 충분한 특징을 사용하지 않는다면, 더 많은 데이터를 제공해도 도움이 되지 않습니다.
- 이는 모델의 복잡성이 부족하여 데이터의 패턴을 제대로 학습하지 못하는 경우를 의미합니다.

ROC와 AUC

- ROC = 수신자 조작 특성 곡선(Receiver Operating Characteristic)
- AUC = 곡선 아래 면적(Area Under Curve)

- ROC 곡선은 모든 가능한 임계값(threshold)에서 분류 모델의 성능을 보여주는 그래프입니다.
- 임계값은 특정 값을 초과할 때 특정 클래스에 속한다고 판단하는 값을 의미합니다.
- 이 곡선은 두 가지 매개변수 사이의 관계를 나타냅니다:
- **TRUE POSITIVE RATE (TPR)**: 진짜 양성 비율
- **FALSE POSITIVE RATE (FPR)**: 가짜 양성 비율

- 기본적으로 TPR/Recall/민감도는 올바르게 식별된 양성 예제의 비율이며, FPR은 잘못 분류된 음성 예제의 비율입니다.
- 앞서 말했듯이 ROC는 가능한 모든 임계값에 대한 TPR과 FPR 간의 관계를 나타내는 곡선이며, AUC는 이 ROC 곡선 아래의 전체 면적입니다.

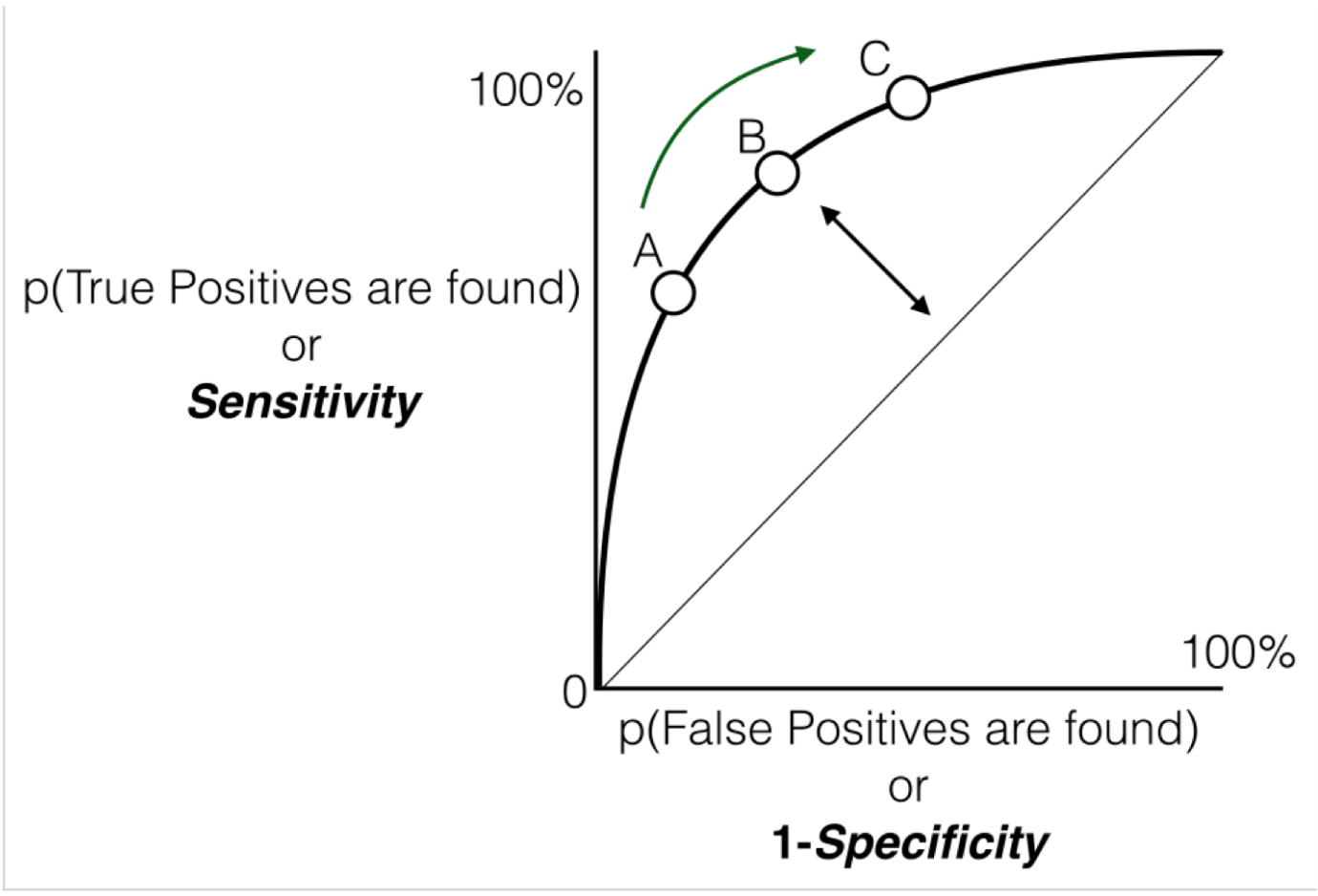
최적의 예측 방법

- 최적의 예측 방법은 ROC 공간의 좌상단 구석 또는 좌표 (0,1)에 해당하는 점을 산출해야 합니다.
- 이는 100% 민감도(거짓 음성이 없음)와 100% 특이도(거짓 양성이 없음)를 나타냅니다.
- (0,1) 점은 완벽한 분류(perfect classification)라고도 합니다.
- 무작위 추측은 대각선(소위 '무차별선')을 따라 좌하단에서 우상단 구석까지의 점을 제공합니다.
- 직관적인 예로는 동전 던지기로 결정하는 것을 들 수 있습니다.
- 표본 크기가 커질수록 무작위 분류기의 ROC 점은 대각선으로 수렴합니다. 균형 잡힌 동전의 경우 (0.5, 0.5) 점으로 수렴합니다.
- 대각선은 ROC 공간을 나눕니다. 대각선 위의 점들은 무작위보다 좋은 분류 결과를 나타내며, 대각선 아래의 점들은 무작위보다 나쁜 결과를 나타냅니다.
- 일관되게 나쁜 예측자의 출력은 반대로 뒤집어 좋은 예측자를 얻을 수 있습니다.

요약

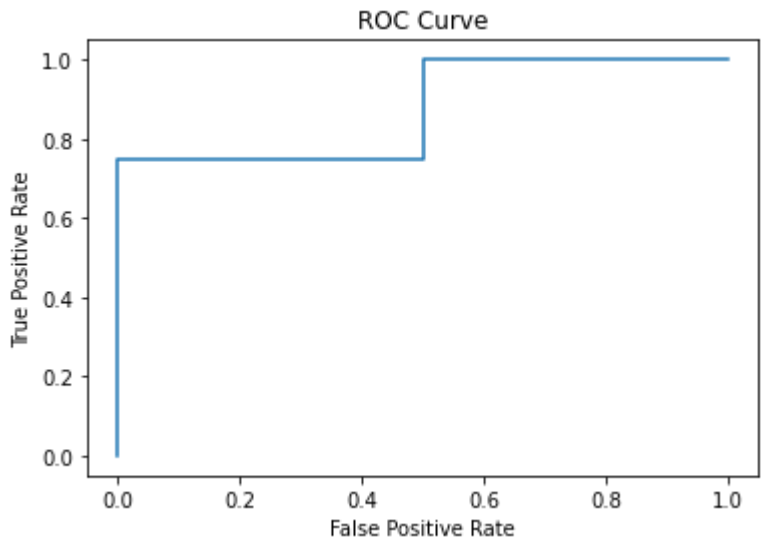
- **ROC 곡선**: 가능한 모든 임계값에 대해 모델의 TPR(진짜 양성 비율)과 FPR(가짜 양성 비율) 간의 관계를 나타내는 그래프.
- **AUC**: ROC 곡선 아래의 전체 면적을 나타내며, 모델의 분류 성능을 종합적으로 평가하는 지표.
- **최적의 예측**: ROC 곡선의 좌상단 (0,1) 점을 나타내며, 이는 완벽한 분류를 의미합니다.
- **무작위 추측**: 대각선을 따라 분포하며, 이는 모델이 무작위로 예측하는 경우를 나타냅니다.

> 💡 핵심.
>
> ROC 곡선과 AUC는 분류 모델의 성능을 평가하는 중요한 도구로, 모델의 예측 성능을 시각적으로 비교하고 분석할 수 있습니다.



History

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve

# 데이터의 실제 레이블
y_true = [0, 0, 1, 1, 1, 1]

# 예측된 점수
y_scores = [0.1, 0.4, 0.35, 0.8, 0.9, 0.7]

# 거짓 양성 비율(FPR)과 진짜 양성 비율(TPR) 계산
fpr, tpr, thresholds = roc_curve(y_true, y_scores)

# ROC 곡선 그리기
plt.plot(fpr, tpr)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()
```



```
print(list(zip(fpr, tpr, thresholds)))
# [(0.0, 0.0, 1.9), (0.0, 0.25, 0.9), (0.0, 0.75, 0.7), (0.5, 0.75, 0.4), (0.5, 1.0, 0.35), (1.0
```

```
from sklearn .metrics import roc_auc_score

auc = np.round(roc_auc_score(y_true, y_pred), 3)
print("Auc for our sample data is {}". format(auc))

# Auc for our sample data is 0.875
```

Feature Extraction and Selection (특징 추출 및 선택)

- 데이터에 특징(feature)이 충분하지 않으면 모델이 과소적합(underfit)될 가능성이 높습니다.
- 반면, 데이터에 너무 많은 특징이 있으면 과적합(overfit)되기 쉽습니다.
- 그렇다면 특징이란 무엇이며, 어디에서 오는 것일까요?
- 특징은 우리가 모델에 제공하는 입력값입니다.
- 가장 간단한 경우, 특징은 단순히 주어집니다.

유용한 도구들:

- **차원 축소(Dimensionality Reduction)**:
  - 데이터의 차원을 줄여서 중요한 특징만 남기는 방법입니다.
  - 예시: PCA(주성분 분석), t-SNE
- **정규화(Regularization)**:
  - 더 많은 특징을 사용하는 모델에 패널티를 부과하는 방법입니다.
  - 예시: SVM(서포트 벡터 머신), 회귀 분석(regression)
- **드롭아웃(Dropout)**:
  - 딥 러닝에서 과적합을 방지하기 위해 일부 뉴런을 무작위로 제외하는 방법입니다.

Occam's Razor (오컴의 면도날)

- 간단한 모델이 더 나은 성능을 발휘하는 경향이 있다는 원칙입니다.
- 가능한 한 적은 수의 특징으로 모델을 만드는 것이 좋습니다.

스팸 필터 예제

- 스팸 필터를 구축하여 이메일이 스팸인지 아닌지를 예측한다고 가정해봅시다.
- 대부분의 모델은 단순히 텍스트로 이루어진 원본 이메일을 처리할 수 없습니다.
- 따라서 특징(feature)을 추출해야 합니다.
- 예를 들어:
  - 이메일에 "Viagra"라는 단어가 포함되어 있는가?
  - 문자 'd'가 몇 번 나타나는가?
  - 발신자의 도메인은 무엇인가?

특징 선택 방법

- **경험과 도메인 전문 지식**이 중요한 역할을 합니다.
- 많은 이메일을 받아봤다면 특정 단어의 존재가 스팸 여부를 잘 나타낼 수 있다는 감을 가질 수 있습니다.
- 반면, 문자 'd'의 개수는 스팸 여부를 잘 나타내지 못할 가능성이 높다는 것도 알 수 있습니다.

- **그러나 일반적으로는 다양한 시도를 해보아야 하며, 이것이 특징 선택의 재미 중 하나입니다.**