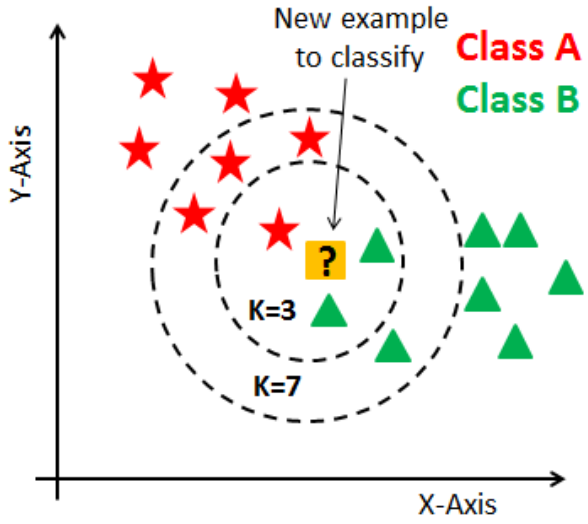
1. KNN

원리

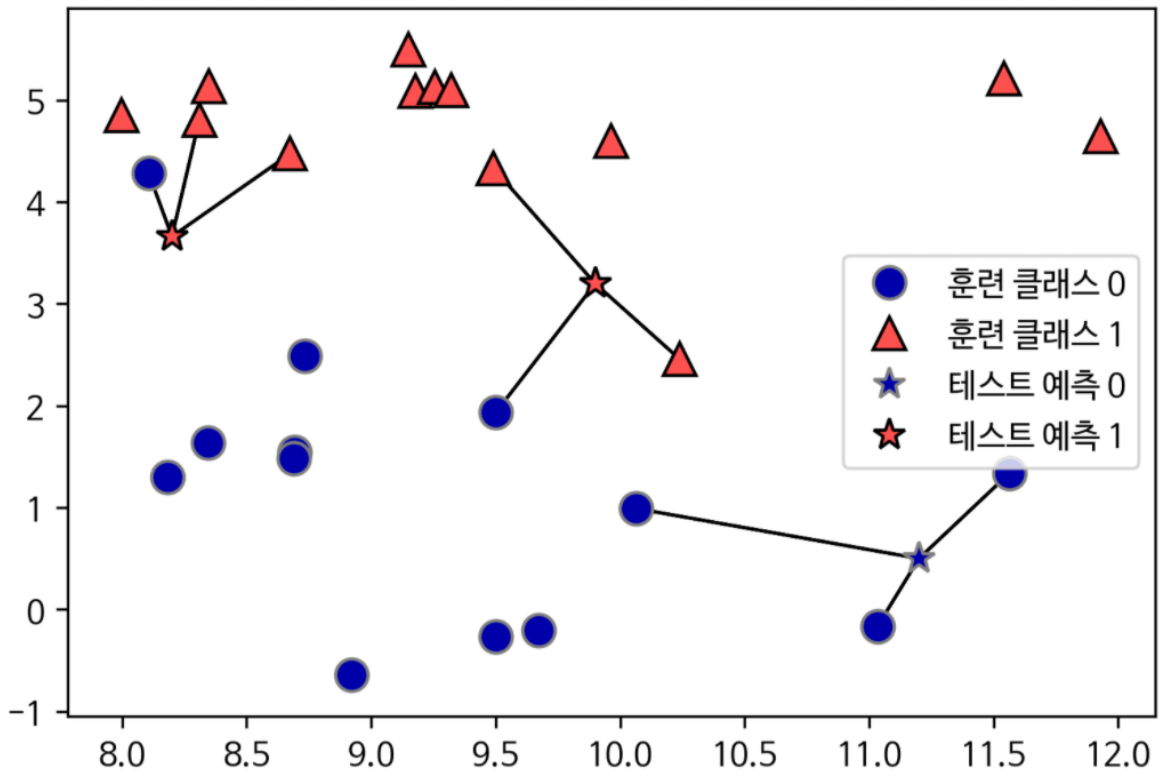
- 서로 가까운 점들은 유사하다는 가정을 가지는 알고리즘
- KNN 알고리즘은 훈련 데이터셋에서 새로운 데이터 포인트와 가장 가까운 훈련 데이터 포인트(최근접 이웃)를 찾아 이를 통해 예측을 진행



종류

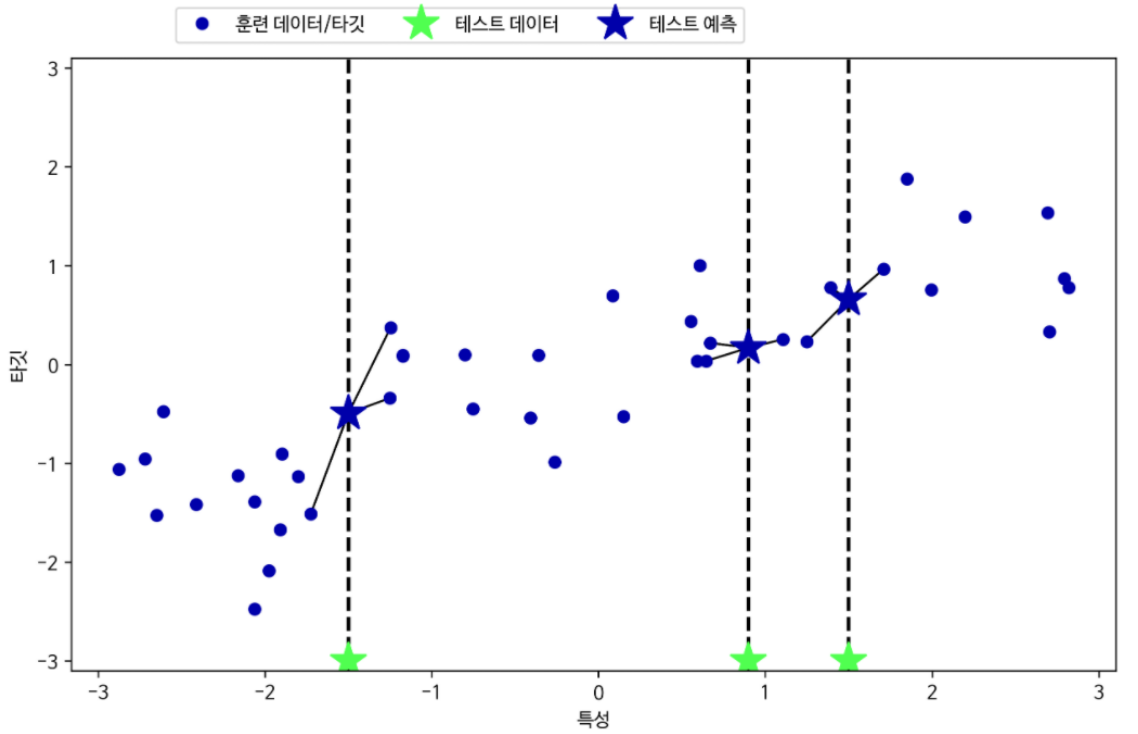
- K-최근접 이웃 분류
  - 새로운 데이터 포인트와 가까운 훈련 데이터 포인트 K개 중 가장 빈도가 높은 것으로 새로운 데이터 포인트를 분류함
    - 단, 공동 1등이 생긴다면 3가지 조치를 취할 수 있음
      - 여러 1등 중 임의로 하나를 정함
      - 거리를 가중치로 사용해서 거리 기반 투표를 함
      - 단독 1등이 생길 때 까지 K를 하나씩 줄임

```
mglearn.plots.plot_knn_classification(n_neighbors=3)
```



- K-최근접 이웃 회귀
  - 새로운 데이터 포인트와 가까운 훈련 데이터 포인트 K개의 평균으로 새로운 데이터 포인트 값을 예측함

```
mglearn.plots.plot_knn_regression(n_neighbors=3)
```



매개변수

- 데이터 포인트 사이의 거리를 재는 방법(= 유사도 측정 방법)
  - 유클리디언 거리(Euclidean distance)
    - 각 속성들 간의 차이를 모두 고려하여 계산
    - 계산값이 0에 가까울수록 유사한 것

$$d(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$$

$$\text{where } x = (x_1, x_2, \ldots, x_n), y = (y_1, y_2, \ldots, y_n)$$

  - 마할라노비스 거리(Mahalanobis distance)
    - 유클리디언 거리에서 데이터의 속성들의 공분산(covariance)을 반영하여 거리를 계산하는 방법
    - 계산값이 0에 가까울수록 유사한 것

$$d(x,y) = \sqrt{(x-y)\Sigma^{-1}(x-y)^T}$$

where $x = (x_1, x_2, \dots, x_n), y = (y_1, y_2, \dots, y_n)$

$$\Sigma = \begin{pmatrix} Cov(X_1, X_1) & \cdots & Cov(X_1, X_n) \\ \vdots & \ddots & \vdots \\ Cov(X_n, X_1) & \cdots & Cov(X_n, X_n) \end{pmatrix}$$

- 민코스키 거리(Minkowski distance)
  - 유클리디언 거리가 각 속성들 간의 차이를 모두 고려한 거리라면, 민코스키 거리는 가장 큰 차이만을 가지고 거리를 계산하는 방법
  - 계산값이 0에 가까울수록 유사한 것

$$d(x,y) = \lim_{p\to\infty}\left[\sum_{i=1}^{n}(x_i - y_i)^p\right]^{\frac{1}{p}} = \max_{1\le i\le n}|x_i - y_i|$$

where $x = (x_1, x_2, \dots, x_n), y = (y_1, y_2, \dots, y_n)$

- 코사인 유사도(Cosine simliarity)
  - 각(radian) 기반의 계산법
  - 벡터의 크기에 영향을 받지 않는 특징
  - 값의 범위는 -1~1이며, 1에 가까울수록 유사한 것

$$sim(x,y) = \frac{\langle x, y\rangle}{\|x\|\|y\|}$$

where $x = (x_1, x_2, \dots, x_n), y = (y_1, y_2, \dots, y_n)$

$$\|x\| = \sqrt{\sum_{i=1}^{n}(x_i)^2}, \langle x, y\rangle = \sum_{i=1}^{n} x_i y_i$$

- 이웃의 수(K)
  - 3개나 5개 정도로 적을 때 잘 작동함

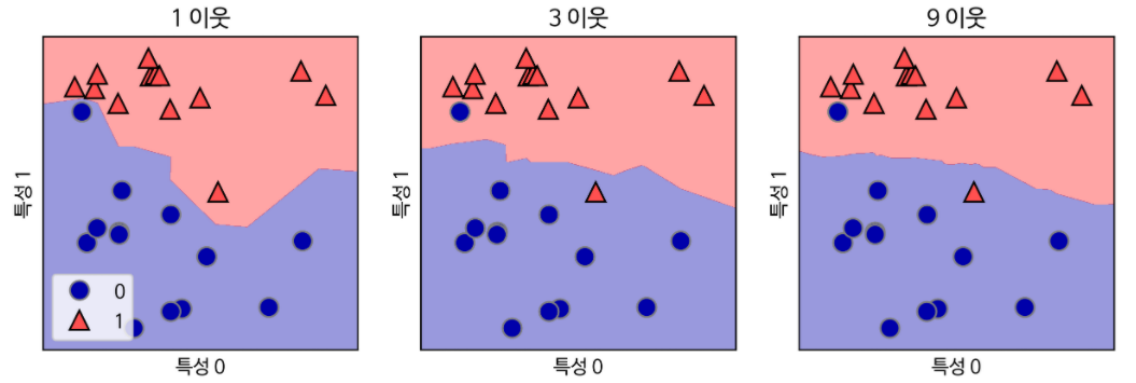
- 이웃의 수가 커지면 결정 경계는 부드러워지고(단순한 모델), 이웃의 수가 작아지면 결정경계는 훈련 데이터에 가깝게 됨(복잡한 모델)

KNeighborsClassifier 분석

```
fig, axes = plt.subplots(1, 3, figsize=(10, 3))

for n_neighbors, ax in zip([1, 3, 9], axes):
    # fit 메소드는 self 오브젝트를 리턴합니다
    # 그래서 객체 생성과 fit 메소드를 한 줄에 쓸 수 있습니다
    clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=True, eps=0.5, ax=ax, alpha=.4)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{} 이웃".format(n_neighbors))
    ax.set_xlabel("특성 0")
    ax.set_ylabel("특성 1")
axes[0].legend(loc=3)
```

```
<matplotlib.legend.Legend at 0x7fd2b4d14b38>
```

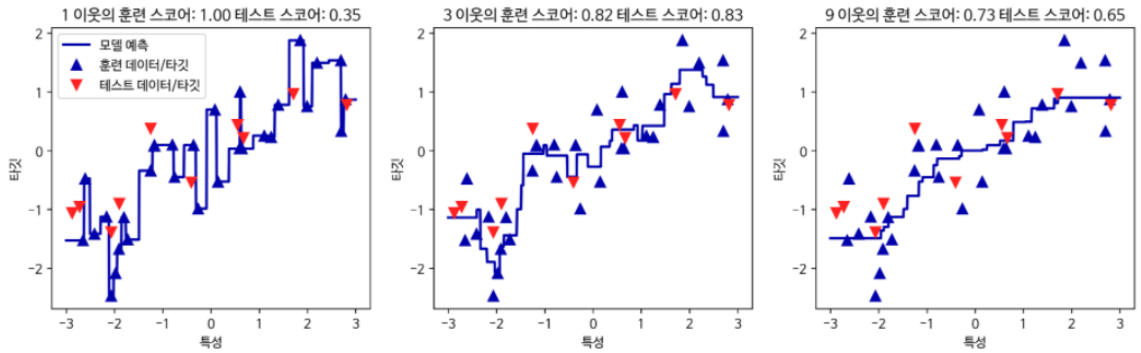


KNeighborsRegressor 분석

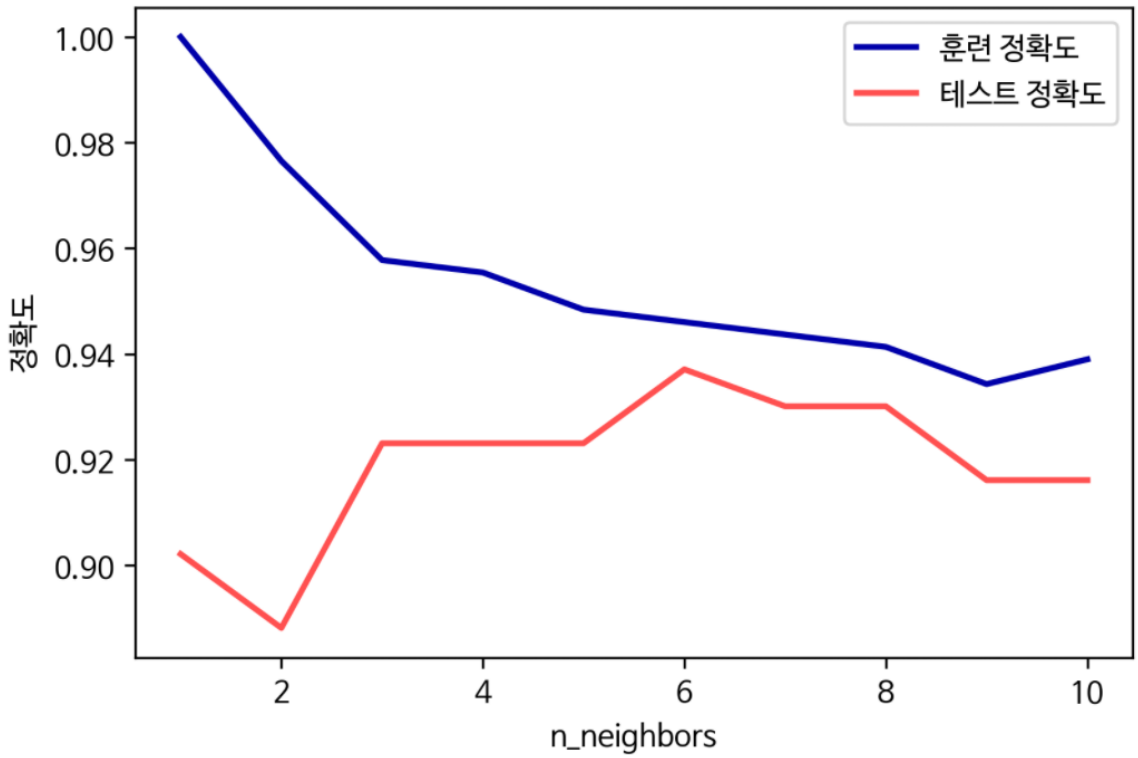
```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))
# -3 과 3 사이에 1,000 개의 데이터 포인트를 만듭니다
line = np.linspace(-3, 3, 1000).reshape(-1, 1)
for n_neighbors, ax in zip([1, 3, 9], axes):
    # 1, 3, 9 이웃을 사용한 예측을 합니다
    reg = KNeighborsRegressor(n_neighbors=n_neighbors)
    reg.fit(X_train, y_train)
    ax.plot(line, reg.predict(line))
    ax.plot(X_train, y_train, '^', c=mglearn.cm2(0), markersize=8)
    ax.plot(X_test, y_test, 'v', c=mglearn.cm2(1), markersize=8)

    ax.set_title(
        "{} 이웃의 훈련 스코어: {:.2f} 테스트 스코어: {:.2f}".format(
            n_neighbors, reg.score(X_train, y_train), reg.score(X_test, y_test)))
    ax.set_xlabel("특성")
    ax.set_ylabel("타깃")
axes[0].legend(["모델 예측", "훈련 데이터/타깃", "테스트 데이터/타깃"], loc="best")
```

```
<matplotlib.legend.Legend at 0x7fd2b47794a8>
```



- 훈련 데이터 전체 개수를 이웃의 수로 지정하는 극단적인 경우, 모든 테스트 포인트가 같은 이웃을 가지게 되므로 테스트 포인트에 대한 예측값은 모두 같은 값이 됨(=즉, 훈련 세트에서 가장 많은 데이터 포인트를 가진 클래스가 예측값이 됨). 그래서 k가 너무 작아도 안되고 너무 커도 문제가 있기 때문에 적당한 K의 개수를 구할 필요가 있음



장점

- 이해하기 쉬운 모델
- 매개변수를 많이 조정하지 않아도 자주 좋은 성능을 발휘함
- 더 복잡한 알고리즘을 적용해보기 전에 시도해볼 수 있음(=베이스 라인)
- 모델 훈련 시간이 필요 없어서 매우 빠르게 모델 구축이 가능함

단점

- 데이터 전처리 과정이 매우 중요
  - 수치형 데이터들의 값을 같은 범위로 맞춰주어야 함
    - 정규화(normalization)

      ```
      normalized = (x-min(x))/(max(x)-min(x))
      ```

    - 표준화(standardization)

```
standardized = (x-mean(x))/std(x)
```

- 명목/범주형 데이터의 경우 one hot encoding을 사용해 더미 변수로 만들어줘야 함

	A	B	C	D	E	F	G	H	I
1	Original data:			One-hot encoding format:					
2	id	Color		id	White	Red	Black	Purple	Gold
3	1	White		1	1	0	0	0	0
4	2	Red		2	0	1	0	0	0
5	3	Black		3	0	0	1	0	0
6	4	Purple		4	0	0	0	1	0
7	5	Gold		5	0	0	0	0	1
8									

- 훈련 세트(특성의 수, 샘플의 수)가 너무 크면 예측이 느려짐
- 수백 개 이상의 많은 특성을 가진 데이터셋에는 잘 동작하지 않음
  - 차원의 저주
    - 두 점이 가깝다라고 하려면 모든 차원에 대해 가까워야 하기 때문에, 차원이 추가된다는 것은 두 점이 가까울 수 있는 가능성이 현저히 줄어든다는 것을 의미함
    - 고차원일 때는 근접이웃들이 평균 거리와 큰 차이가 나지 않게 되고, 그렇기 때문에 가깝다는 것이 별 의미를 가지지 않게 됨
    - 따라서, 고차원에서 KNN을 이용하려면 먼저 차원 축소를 하는 것이 좋음
- 특성 값이 대부분 0인, 희소한 데이터셋과는 특히 잘 작동하지 않음

참고

- [블로그]유사도 측정 방법
  - https://m.blog.naver.com/PostView.nhn?blogId=cjh226&logNo=220810613028&proxyReferer=https:%2F%2Fwww.google.com%2F
- [책]파이썬 라이브러리를 활용한 머신러닝
- [책]밑바다부터 시작하는 데이터 과학