아래 강의 노트는 [Python Data Science Handbook (http://shop.oreilly.com/product/0636920034919.do)](http://shop.oreilly.com/product/0636920034919.do) 5장을 기반으로 번역 및 편집하여 페이지 구성함

초모수와 모델 검증

앞 장에서 지도 학습 모델을 적용하는 기본 단계를 소개 했다.:

1. 모델 클래스 선택
2. 모델 초모수 선택
3. 모델을 훈련 데이터에 적합
4. 모델을 사용해 새 데이터에 대한 레이블 예측

모델 검증에 대한 고려 사항

모델 검증은 매우 단순하다. 모델과 모델의 초모수를 선택한 후, 훈련 데이터 일부에 이를 적용하고 예측 값을 알려진 값과 비교해서 이 모델이 얼마나 효과적인지 추정 할 수 있다.

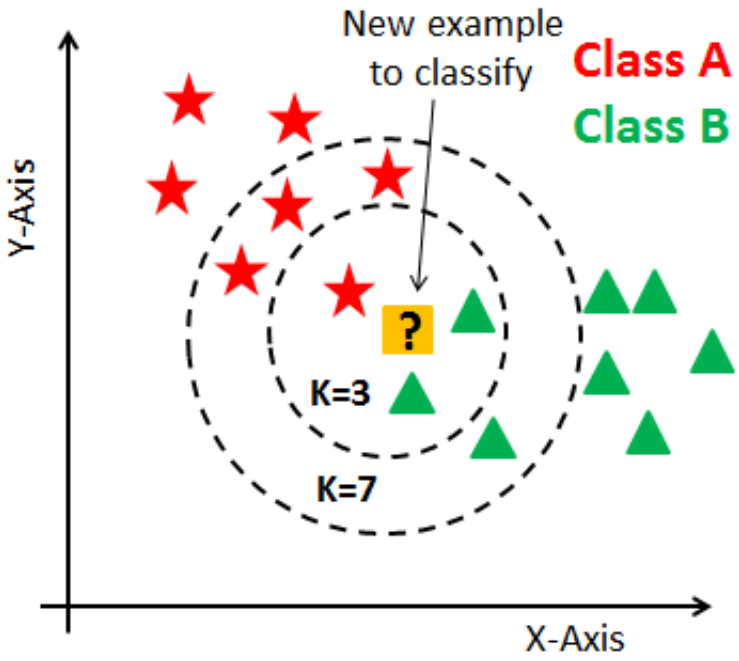
견고한 모델 평가를 위해서 검정 표본과 교차 검증을 사용 하는 방법을 살펴보고자 한다.

잘못된 방식의 모델 검증

데이터 로딩:

In [1]:
```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
```

모델과 초모수를 선택하는 n_neighbors=1 인 k이웃 분류기를 사용. [



] 알려지지 않은 점의 레이블은 훈련 데이터 중 그와 가장 가까운 점의 레이블과 같다 라는 매우 간단하고 직관적인 모델

In [2]:
```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=1)
```

모델을 훈련 시기코 그 모델을 사용해 이미 알고 있는 데이터의 레이블을 예측:

In [3]:
```
model.fit(X, y)
y_model = model.predict(X)
```

마지막으로 올바른 레이블을 가진 점의 비율을 계산

In [4]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y, y_model)
```

Out[4]: 1.0

정확도가 1인데 이는 모델을 사용해 100% 모든 점이 올바른 레이블을 갖게 된다는 뜻이다. 하지만 기대 정확도를 정말로 측정하고 있는 것일까?

이 접근 방식은 같은 데이터로 모델을 훈련하고 검증한다는 근본적인 오류를 내포하고 있다. 게다가 최근접 이웃모델은 단순히 훈련데이터를 저장 하고 이 데이터들과 새로운 데이터를 데이터를 비교해 레이블을 예측하는 인스턴스 기반의 추정모델이기 때문에 억지스러운 경우를 제외하고 항상 100%의 정확도를 갖는다.

올바른 방식의 모델 검증: 검정 표본

holdout set (검정표본) 방식을 사용하여 모델의 훈련 데이터에서 데이터의 일부를 빼내 그것은 모델 성능을 확인하는 검정 표본으로 사용한다. Scikit-Learn에서는 `train_test_split` 유틸리티를 사용하면된다:

In [5]:
```
from sklearn.model_selection import train_test_split
# split the data with 50% in each set
X1, X2, y1, y2 = train_test_split(X, y, random_state=0,
                                  train_size=0.5)

# fit the model on one set of data
model.fit(X1, y1)

# evaluate the model on the second set of data
y2_model = model.predict(X2)
accuracy_score(y2, y2_model)
```

Out[5]: 0.9066666666666666

교차 검증을 통한 모델 검증

모델 검증에 검정 표본을 사용할 때의 한 가지 단점은 모델을 훈련시킬 데이터의 일부를 읽게된다는 것이다. 앞의 경우에 데이터세트의 반은 모델을 훈련 시키는데 기여하지 못한다. 특히 초기 데이터가 작은 경우에는 적절하지 않은 방법이다.

이러한 문제를 해결하는 한가지 방법은 *cross-validation*을 사용하는 것이다. 즉 데이터의 각 하위 집합이 훈련자료와 검증 자료로 사용되도록 일련의 적합을 수행하는 것이다.

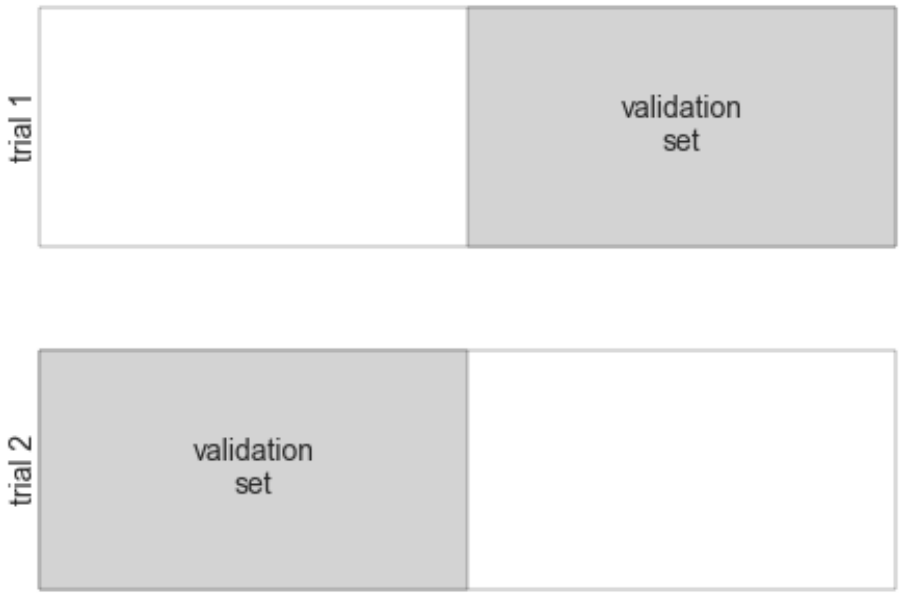


figure source in Appendix (06.00-Figure-Code.ipynb#2-Fold-Cross-Validation)

두 건의 검증을 시행하는데, 각 검증 작업에서 데이터의 반을 검정 표본으로 사용한다. 이전의 분할 데이터를 사용해서 다음과 같이 구현 할 수 있다.

In [6]:
```
y2_model = model.fit(X1, y1).predict(X2)
y1_model = model.fit(X2, y2).predict(X1)
accuracy_score(y1, y1_model), accuracy_score(y2, y2_model)
```

Out[6]: (0.96, 0.9066666666666666)

그 결과로 두개의 정확도를 얻게되는데 전체 모델 성능을 더 잘 측정하기 위해서 이 두 정확도를 결합할 수 있다. 이 특정 형태의 교차 검증을 2겹 교차 검증이라 한다. 데이터를 두 집합으로 나눠 각 검정 시행에 차례대로 검정 표본으로 사용한다.:

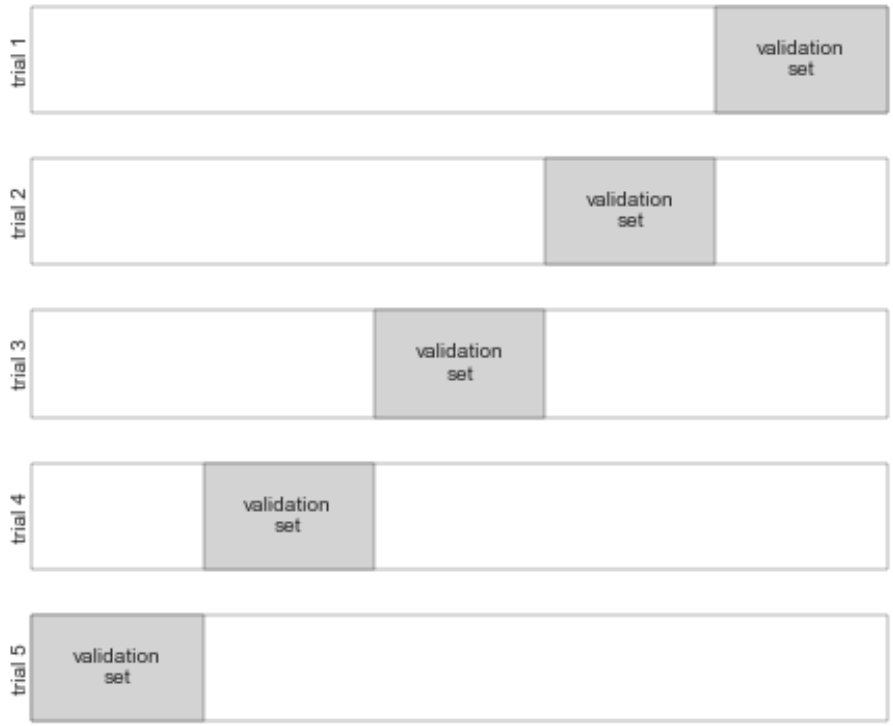


figure source in Appendix (06.00-Figure-Code.ipynb#5-Fold-Cross-Validation)

이 개념을 확장해 더 많은 시행과 더 많은 겹의 데이터를 사용할 수도 있다. 위의 그림에서는 다섯 그룹으로 나누어 검증을 시행할때마다 차례대로 하나씩 모델 평가를 위한 검정 표번으로 사용하고 나머지 4/5 데이터는 모델의 적합시키는데 사용한다. Scikit-Learn의 `cross_val_score` 편의 루틴을 이용해 간결하게 수행 할 수 있다. :

In [7]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(model, X, y, cv=5)
```

Out[7]:
```
array([0.96666667, 0.96666667, 0.93333333, 0.93333333, 1.        ])
```

Scikit-Learn 은 특정 상황에 유용한 수많은 교차 검증 방식을 구현 한다. 이 방식은 `cross_validation` 모듈의 반복자를 통해 구현 된다. 예를 들어 데이터 점의 개수와 같은 수 만큼 반복하는 교차 검증을 하는 극단적인 경우를 원할 수 도 있다. 따라서 검증을 시행할 때 마다 한 점을 제외한 모든 점에 대해 훈련하는 방법을 *leave-one-out* 교차검증이라 부르며 다음과 같이 사용할 수 있다.

In [8]:
```
from sklearn.model_selection import LeaveOneOut
scores = cross_val_score(model, X, y, cv=LeaveOneOut())
scores
```

Out[8]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

150 개의 표본을 가지기 때문에 단일 관측지 제거 방식의 교차검증이 150건의 시행에 대한 점수를 생산하고 그 점수는 예측 성공(1.0) 또는 실패 (0.0)를 나타낸다. 이 점수들을 평균을 계산해서 오류율을 추정한다.

In [9]:
```
scores.mean()
```

Out[9]:
```
0.96
```

최적의 모델 선택하기

검증과 교차 검증의 기본 내용을 살펴 보았으니 이제 모델과 초모수 선택과 관련해 더 깊이 알아 본다. 다음과 같은 질문을 하는 것이 가장 중요하다. 추정기 성과가 저조하다면 어떻게 개선을 할 것인가? 이 문제를 해결하는 몇가지 답이 있다.

- 더 복잡하거나 더 유연한 모델 사용
- 덜 복잡하거나 덜 유연한 모델 사용
- 더 많은 훈련 표본 수집
- 각 표본에 특징을 추가하기 위해 더 많은 데이터 수집

편향 분석 트레이드오프

근본적으로 최고의 모델을 선택하는 것은 편향과 분산 사이의 트레이드오프에서 가장 효율적인 점을 찾는 것이다. 동일한 데이터세트에서 두개의 회귀 모델을 나타내는 그림을 보자

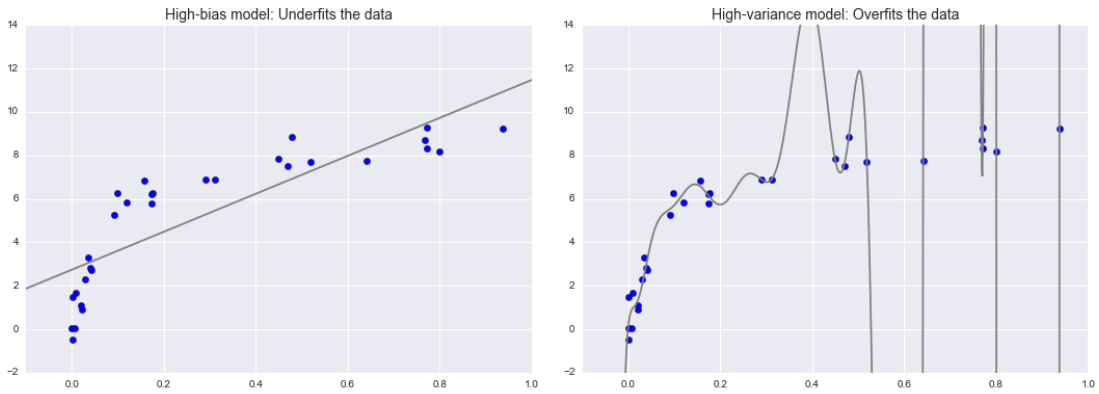


figure source in Appendix (06.00-Figure-Code.ipynb#Bias-Variance-Tradeoff)

이 모델 중 어느 것도 데이터에 특별히 잘 적합하지는 않지만 이 둘은 서로 방식으로 실패한다.

위 그림의 오른쪽 모델은 고차 다항식을 데이터에 적합시키려고 한다. 이 모델 적합은 데이터의 세밀한 특징까지 거의 완벽히 설명 할 수 있는 충분한 유연성을 가지고 있지만, 그것이 훈련 데이터를 아주 정확하게 설명하고 있더라도 그 형태는 해당 데이터를 생성한 프로세스의 내재된 속성보다 데이터의 특정 잡음 속성을 더 많이 반영하는 것 처럼 보인다.이러한 모델을 데이터를 과적합 한다고 말한다. 즉 모델이 기본 데이터 분포와 함께 임의의 오류까지 설명할 정도로 너무 과한 모델 유연성을 가지고 있다. 다른 말로 하면 이모델은 고분산 모델을 가지고 있다.

.

이것을 다른 관점에서 보기 위해 새로운 데이터에 대한 y값을 예측 하는데 이 두모델을 사용하면 어떤일이 발생할지 생각해 본다. 아래 그림에서 밝은 빨간색 점은 훈련자료에서 생략된 데이터를 나타낸다

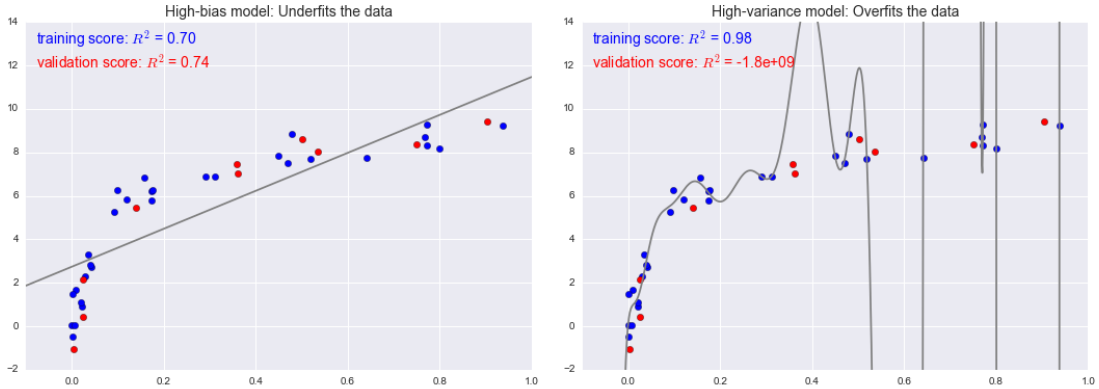


figure source in Appendix (06.00-Figure-Code.ipynb#Bias-Variance-Tradeoff-Metrics)

여기서 점수는 모델이 대상값의 단순 평균에 비해 얼마나 잘 수행하는지 측정하는 점수 또는 결정계수를 말한다. coefficient of determination (https://en.wikipedia.org/wiki/Coefficient_of_determination), $R^2 = 1$ 완전히 일치 함을 나타내고, $R^2 = 0$은 모델이 데이터의 단순 평균을 구하는 수준에 지나치지 않으며 음수는 그보다 더 나쁜 모델이라는 것을 의미한다. 이 두모델과 관련된 점수로 모델 성능에 대해 더 일반적으로 나타낼 수 있다.

- 고편향 모델의 경우, 검정 표본에서의 모델 성능이 훈련 표본에서의 성능과 유사하다.
- 고분산 모델의 경우 검정 표본에서의 모델 성능이 훈련 표본에서 성능 보다 훨씬 더 떨어진다.

모델의 복잡도를 조정할 수 있다고 상상하면 훈련 표본 점수와 검정 표본 점수가 아래와 같이 움직 인다.

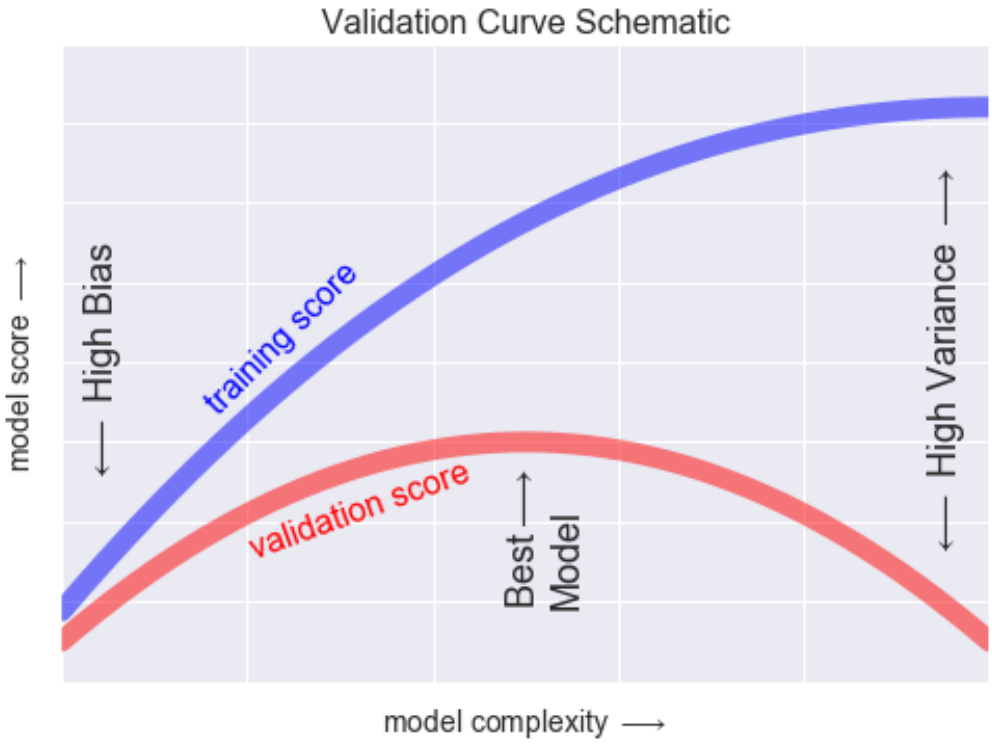


figure source in Appendix (06.00-Figure-Code.ipynb#Validation-Curve)

위 그림을 검증 곡선이라고 하며 다음과 같은 근본적인 특징을 알 수 있다.

- 훈련 점수는 언제가 검증 점수보다 높다. 모델은 아직 보지 못한 데이터보다 이미 본 데이터에 더 잘 적합하기 때문에 이것은 당연한 결과라고 할 수 있다.
- 모델 복잡도가 너무 낮은 경우 (고편향 모델)훈련 데이터가 과소적합되는데 이는 모델의 훈련 데이터와 미리 보지 않은 데이터 모두를 예측 하지 못한다는 뜻이다.
- 모델 복잡도가 너무 높은 경우 (고분산 모델) 훈련데이터가 과적합되는데 이것은 모델이 훈련 데이터는 매우 잘 예측 하지만 본 적이 없는 데이터에 대해서는 예측에 실패한다는 뜻이다.
- 중간값에서 검증 곡선은 최댓값을 가진다. 복잡도가 이 수준이라는 것은 편향과 분산 사이의 적절한 트레이드 오프가 이뤄졌음을 나타낸다. The means of tuning the model complexity varies from model to model; when we discuss individual models in depth in later sections, we will see how each model allows for such tuning.

Scikit-Learn 검증 곡선

다향 회귀모델을 사용해 검증곡선을 살펴보고자 한다. 이 모델은 일반화된 선형 모델로 조정 가능한 매개변수는 다항식으 차수이다. 예를 들어 1차식은 데이터에 직선을 적합시키르모 모델의 모수 a 와 b에 대해 다음과 같이 기술 할 수 있다.:

$$y = ax + b$$

3차 다항식은 3차 곡선 데이터에 적합 시킨다. 모델 모수 a, b, c, d 에 대해 다음과 같이 기술 할 수 있다. :

$$y = ax^3 + bx^2 + cx + d$$

이를 어떤 차수의 다항식 특징에도 일반화 시킬수 있다. Scikit-Learn에서는 이것을 다항식 전처리 프로그램과 결합한 간단한 선형 회귀로 구현 할 수 있다. 아래 예제는 연산을 하나로 묶는 파이프라인 (https://skasha.tistory.com/80)을 사용할 것이다.

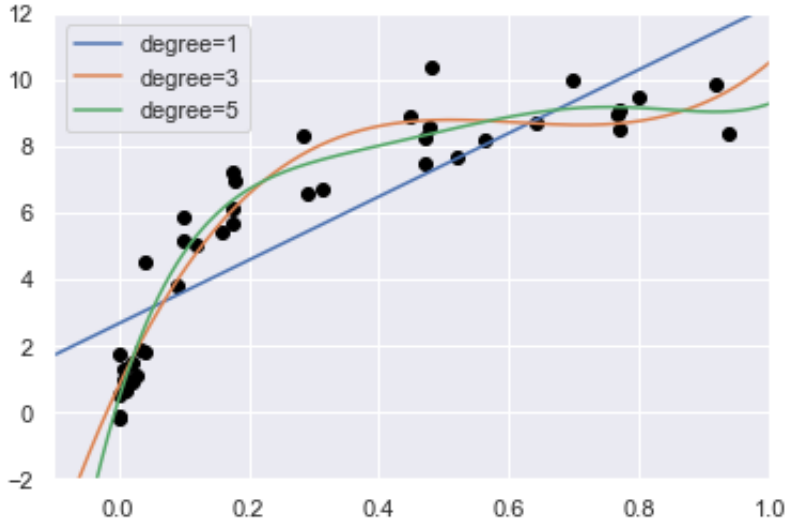
In [10]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

def PolynomialRegression(degree=2, **kwargs):
    return make_pipeline(PolynomialFeatures(degree),
                         LinearRegression(**kwargs))
```

모델이 적합한 데이터를 만들자

In [11]:
```
import numpy as np

def make_data(N, err=1.0, rseed=1):
    # randomly sample the data
    rng = np.random.RandomState(rseed)
    X = rng.rand(N, 1) ** 2
    y = 10 - 1. / (X.ravel() + 0.1)
    if err > 0:
        y += err * rng.randn(N)
    return X, y

X, y = make_data(40)
```

이제 데이터와 함께 여러 차수의 다항식을 적합을 시각화 한다.

In [12]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn; seaborn.set()  # plot formatting

X_test = np.linspace(-0.1, 1.1, 500)[:, None]

plt.scatter(X.ravel(), y, color='black')
axis = plt.axis()
for degree in [1, 3, 5]:
    y_test = PolynomialRegression(degree).fit(X, y).predict(X_t
    plt.plot(X_test.ravel(), y_test, label='degree={0}'.format(
plt.xlim(-0.1, 1.0)
plt.ylim(-2, 12)
plt.legend(loc='best');
```

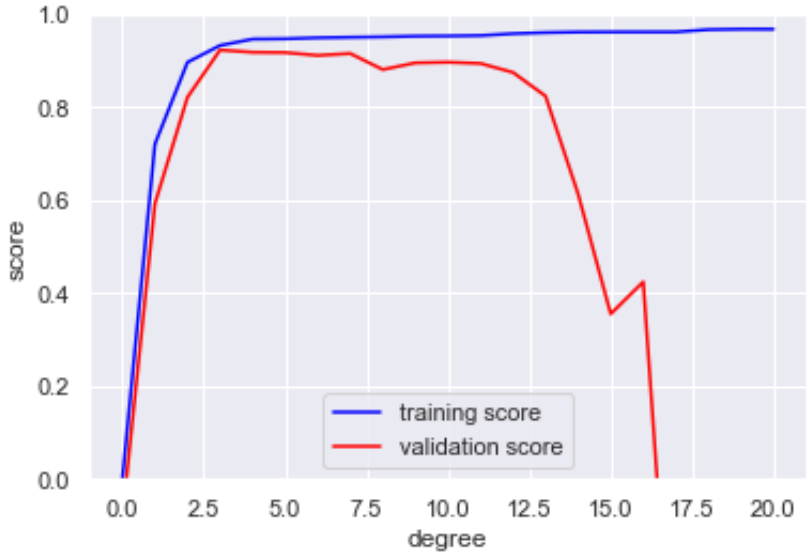


몇 차 다항식이 편향(과소적합)과 분석(과적합)사이에 적절한 트레이드 오프를 제공하는가 ?

이 특정 데이터에 대한 모델에 대한 검증 곡선을 시각화하면 이 질문의 답을 얻을 수 있다. Scikit-Learn이 제공하는 `validation_curve` 편의 루틴을 사용해 간단하게 수행 할 수 있다. 모델과 데이터, 모수 이름, 탐색 범위가 주어지면 이 함수가 자동으로 그 범위 내에서 훈련 점수와 검증 점수를 계산한다.:

In [13]:
```
#from sklearn.learning_curve import validation_curve
from sklearn.model_selection import validation_curve
degree = np.arange(0, 21)
train_score, val_score = validation_curve(PolynomialRegression(
                                          'polynomialfeatures__

plt.plot(degree, np.median(train_score, 1), color='blue', label
plt.plot(degree, np.median(val_score, 1), color='red', label='v
plt.legend(loc='best')
plt.ylim(0, 1)
plt.xlabel('degree')
plt.ylabel('score');
```

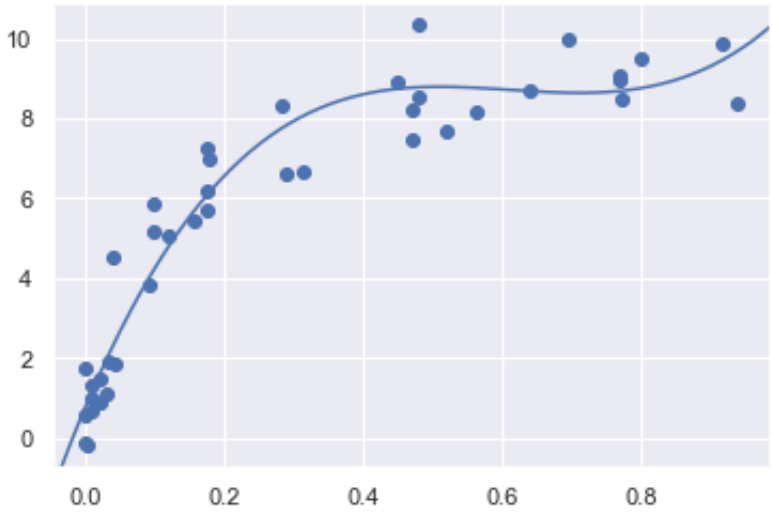
```
/usr/local/lib/python3.8/site-packages/sklearn/utils/validation.py
:67: FutureWarning: Pass param_name=polynomialfeatures__degree, pa
ram_range=[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 1
8 19 20] as keyword args. From version 0.25 passing these as posit
ional arguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```



이 코드는 예상대로 훈련 점수가 언제나 검증 점수보다 높다라는 정성 패턴을 정확히 보여준다. 훈련 점수는 모델 복잡도가 증가함에 따라 단조롭게 향상되고 검증 점수는 모델이 과적합되어 점수가 떨어지기 전에 최댓값에 도달한다.

검증 곡선을 통해 편향과 분산 사이의 최적의 트레이드오프가 3차 다항식임을 알 수 있다. 다음과 같이 원본 데이터에 이 적합을 계싼하고 표시하면된다.:

In [14]:
```
plt.scatter(X.ravel(), y)
lim = plt.axis()
y_test = PolynomialRegression(3).fit(X, y).predict(X_test)
plt.plot(X_test.ravel(), y_test);
plt.axis(lim);
```



최적의 모델을 발견하기 위해 실제로 훈련 점수를 계산할 필요는 없지만 훈련 점수와 같은 검증 점수 사이의 관계를 시험하는 것으로 모델의 성능에 대한 유용한 통찰력을 얻을 수 있다.

< Introducing Scikit-Learn (05.02-Introducing-Scikit-Learn.ipynb) | Contents (Index.ipynb) | Feature Engineering (05.04-Feature-Engineering.ipynb) >

Open in Colab
(https://colab.research.google.com/github/jakevdp/PythonDataScienceHandbook/blob/ma
Hyperparameters-and-Model-Validation.ipynb)