IntroScikitLearn

November 8, 2020

아래 강의 노트는 Python Data Science Handbook 5장을 기반으로 번역 및 편집하여 페이지 구성함

1 Scikit-Learn 소개

파이썬에는 다양한 머신러닝 알고리즘을 견고하게 구현한 라이브러리가 있다. 가장 유명한 라이브러리 중 하나인 Scikit-Learn 패키지는 다양한 일반 알고리즘을 효율적으로 구현해서 제공한다.
Scikit-Learn 은 일관되과 간결한 API와 매우 유용하고 완전한 온라인 문서가 특징이다. 일관성 덕분에 한가지 유형의 모델에 대한 Scikit-Learn기본 구문을 익히고 나면 새로운 모델이나 알고리즘으로 전환하는 것은 매우 간단다.

이점에서 Scikit-Learn에 관해 개괄적으로 살펴본다..

1.1 Scikit-Learn 에서의 데이터 표현 방식

머신러닝은 데이터로부터 모델은 만드는 것과 관련이 있다. 따라서 컴퓨터가 이해할 수 있게 데이터를 표현하는 방식을 먼저 알아본다.

1.1.1 Table로서의 데이터

기본 테이블은 2차원 데이터 그리드로서 행은 데이터 세트의 개별요소를 나타내며 열은 이각 오소와 관련된 수량을 나타내다. 예를 들어 1936년 로널드 피셔(Ronald Fisher)가 분석한 유명한 붓꽃 데이터를 살펴보자 Iris dataset. 데이터는 Seaborn 라이브러리를 사용해 Pandas DataFrame 형태로 다운로드 할 수 있다.:

[1]:
```
import seaborn as sns
iris = sns.load_dataset('iris')
iris.head()
```

[1]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

각 데이터의 행은 하나의 관측된 꽃을 의미하고 행의 수는 데이터세트에 있는 꽃의 전체 개수를 나태니다. 행을 표본이라고 하고 행의 개수를 `n_samples`이라고 한다.

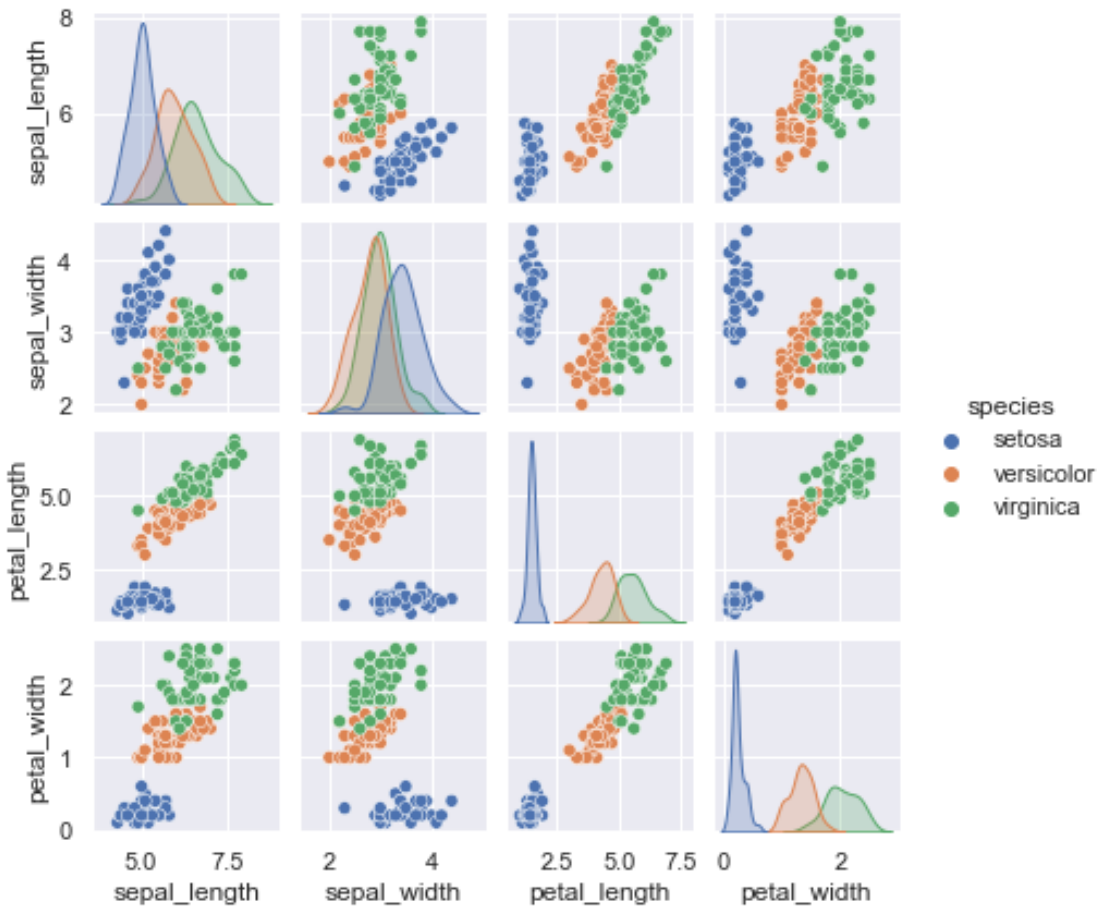
비슷하게 데이터의 각 열은 각 표본을 설명하는 특정 수량 정보를 말한다. 여기서 행렬의 열을 특징 *features* 라고 하고 열의 개수를 `n_features` 이라고 한다.

특징 행렬 테이블 레이아웃은 정보를 2차원 수치 배열이나 행렬로 생각 할 수 있다. 그것을 *features matrix* 이라고 한다. 관례상 특징 행렬을 보통 x라는 변수에 저장된다.

대상행렬 특징 핼령렬 X외에도 관례상 대개 y라고 부를 레이블 또는 대상 배열과도 작업한다. 대상 배열은 대개 길이가 `n_samples`인 1차원 배열이며, NumPy array 나 Pandas `Series`에 포함되는게 일반적이다. 대상 배열은 연속적인 수치나 이산 클래스/ 레이블을 갖을 수도 있다. Seaborn을 사용하면 데이터를 편리하게 시각화 할 수 있다.

[2]:
```
%matplotlib inline
import seaborn as sns; sns.set()
sns.pairplot(iris, hue='species', size=1.5);
```

```
/usr/local/lib/python3.8/site-packages/seaborn/axisgrid.py:1912: UserWarning:
The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



Scikit-Learn을 사용하기 위해 DataFrame연산을 사용해 DataFrame에서 특정 행렬과 대상 배열을 추출한다.

[3]:
```
X_iris = iris.drop('species', axis=1)
X_iris.shape
```

[3]: (150, 4)

[4]:
```
y_iris = iris['species']
y_iris.shape
```

[4]: (150,)

요약 하면 특정값과 대상 값이 예상 레이아웃은 아래와 같다.:

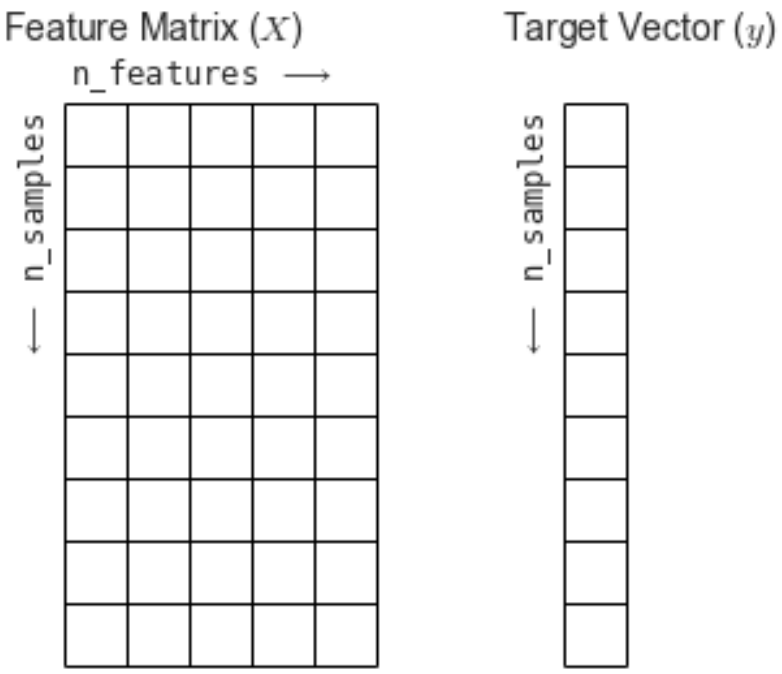


figure source in Appendix

1.1.2 API 기초

일반적으로 Scikit-Learn API를 이용하는 단계는 다음과 같다. .

1. Scikit-Learn으로부터 적절한 추정 클래스를 임포트 해서 모델의 클래스를 선택한다..
2. 이 클래스를 원하는 값으로 인스턴스화 해서 모델의 초모수 (Hyper parameter value)를 선택한다.
3. 데이터를 앞에서 논의 한 내용에 따라 특징 배열과 대상 벡터로 배치한다.
4. 모델 인스턴스의 `fit()` 메서드를 호출해 모델을 데이터에 적합시킨다.
5. 모델의 새 데이터에 적용한다.:
   - 지도 학습인 경우 대체로 `predict()`메서드를 사용해 알려지지 않은 데이터에 대한 레이블을 예측한다.
   - 비지도 학습인 경우 대체로 `transform()` 또는 `predict()` 메서드를 이용해 데이터의 속성을 변환 하거나 추론한다.

https://scikit-learn.org/dev/

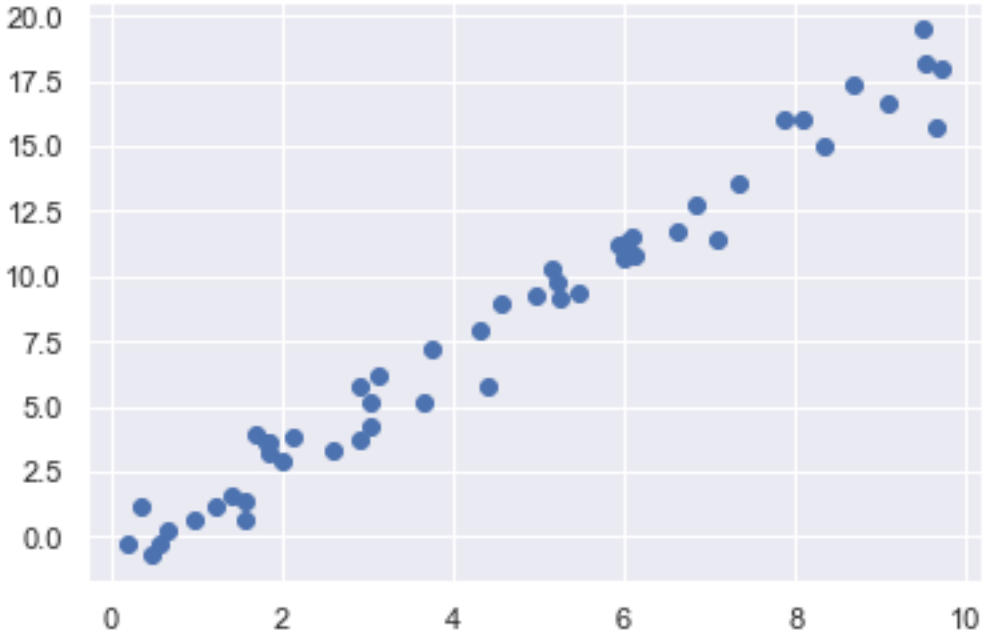
1.1.3 지도 학습 예: 간단한 선형회귀

이 단계를 실행하는 예제로 선을 (x,y)데이터에 적합시키는 일반적인 경우인 간단한 선형 회귀를 생각해 보자 간단한 데이터로 사용:

[5]:
```
import matplotlib.pyplot as plt
import numpy as np

rng = np.random.RandomState(42)
```

```
x = 10 * rng.rand(50)
y = 2 * x - 1 + rng.randn(50)
plt.scatter(x, y);
```



데이터가 준비되면 앞에 설명한 작업단계를 이용할 수 있다.:

1. 모델 클래스를 선택한다. Scikit-Learn에서 모델 클래스는 모두 파이썬 클래스로 표현된다. 따라서 가령 간단한 선형 회귀 모델을 계산하고 싶다면 선형 회귀 클래스를 임포트 하면된다.:

[6]:
```
from sklearn.linear_model import LinearRegression
```

일반적인 회귀모델도 존재하는데 `sklearn.linear_model` module documentation 참고.

2. hyperparameters 선택 중요한 점은 모델 클래스가 모델 인스턴스와 같지 않다.

모델 클래스를 결정했더라도 여전히 몇가지 선택할 옵션이 남아 있다. 작업하는 모델 클래스에 따라 다음 질문 중 하나 이상에 대단해야 한다.

- 오프셋 (i.e., *y*-intercept)에 적합 시킬 것인가?
- 모델을 정규화할 것인가?
- 모델의 유연성을 높이기 위해 특징을 사전 처리 할 것인가?
- 모델의 어느 정도에서 정규화를 사용할 것인가?
- 얼마나 많은 모델 성분을 사용할 것인가?

이것은 모델 클래스가 정해지고 나면 반드시 선택해야 할 중요한 항목이다. 이 선택 사항은 종종 초모수 또는 모델 데이터에 적합시키기 전에 설정돼야 할 모수로 표현된다.

[7]:
```
model = LinearRegression(fit_intercept=True)
model
```

[7]:
```
LinearRegression()
```

모델이 인스턴스화되고 나면 남은 일은 이 초모수 값들은 저장하는 것이다. 여기서 아직 어떤 데이터에도 이 모델을 적용하지 않았다. Scikit-Learn API는 모델을 선정하는 것과 모델을 데이터에 적용하는 것을 명확하게 구분한다.

3. 데이터를 특징 행렬과 대상 벡터로 배치한다. Scikit-Learn에서는 데이터를 표현할 2차원 특징 행렬과 1차원 대상 배열이 필요하다는 점을 자세 하게 설명했다. 대상 변수 y는 이미 적절한 형식 길이(n_samples 배열)인 행렬로 만들어야 한다.

[8]:
```
X = x[:, np.newaxis]
X.shape
y = y.reshape(50,1)
```

4. 모델을 데이터에 적합 시킨다 모델을 데이터에 적용하기 위해fit() 메서드 사용:

[9]:
```
model.fit(X, y)
```

[9]:
```
LinearRegression()
```

fit()명령어에는 모델의 종속과 여러가지 내부 계산이 뒤따르며 계산된 결과는 모델 적용 속성에 지정되어 사용자가 탐색 할 수 있다. Scikit-Learn에 관례상 fit()절차 동안 학습된 모델 모수는 모두 뒤에 밑줄 표시(_) 가 붙는다 :

[10]:
```
model.coef_
```

[10]:
```
array([[1.9776566]])
```

[11]:
```
model.intercept_
```

[11]:
```
array([-0.90331073])
```

5. 알려지지 않은 데이터에 대한 레이블 예측 모델이 훈련되고 나면 지도 학습의 주요 작업은 훈련 데이터에 포함되지 않았던 새 데이터에 대해 이 모델이 무엇이라고 말하는지 기반으로 모델을 평가 하는 것이다. predict() 메서드를 사용해 가능하다:

[12]:
```
xfit = np.linspace(-1, 11)
xfit
```

[12]:
```
array([-1.        , -0.75510204, -0.51020408, -0.26530612, -0.02040816,
        0.2244898 ,  0.46938776,  0.71428571,  0.95918367,  1.20408163,
        1.44897959,  1.69387755,  1.93877551,  2.18367347,  2.42857143,
        2.67346939,  2.91836735,  3.16326531,  3.40816327,  3.65306122,
        3.89795918,  4.14285714,  4.3877551 ,  4.63265306,  4.87755102,
```

```
 5.12244898,  5.36734694,  5.6122449 ,  5.85714286,  6.10204082,
 6.34693878,  6.59183673,  6.83673469,  7.08163265,  7.32653061,
 7.57142857,  7.81632653,  8.06122449,  8.30612245,  8.55102041,
 8.79591837,  9.04081633,  9.28571429,  9.53061224,  9.7755102 ,
10.02040816, 10.26530612, 10.51020408, 10.75510204, 11.        ])
```

x 값을 [n_samples, n_features] 특징 행렬에 맞추어야 하며, 그러고 나면 모델에 그 값을 전달 할 수 있다.:

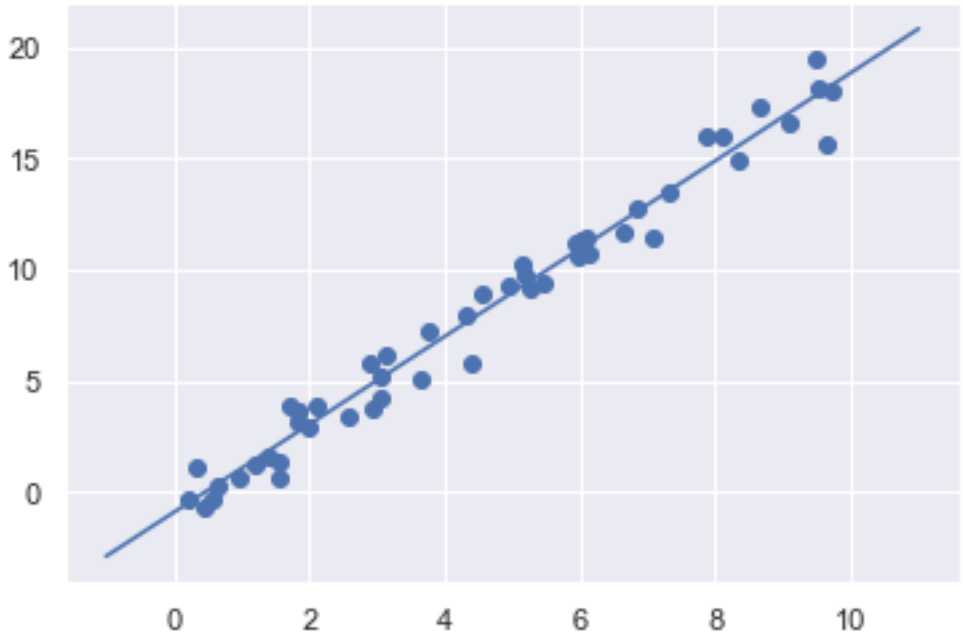
[13]:
```
Xfit = xfit[:, np.newaxis]
yfit = model.predict(Xfit)
yfit
```

[13]:
```
array([[-2.88096733],
       [-2.39664326],
       [-1.9123192 ],
       [-1.42799513],
       [-0.94367106],
       [-0.459347  ],
       [ 0.02497707],
       [ 0.50930113],
       [ 0.9936252 ],
       [ 1.47794926],
       [ 1.96227333],
       [ 2.44659739],
       [ 2.93092146],
       [ 3.41524552],
       [ 3.89956959],
       [ 4.38389366],
       [ 4.86821772],
       [ 5.35254179],
       [ 5.83686585],
       [ 6.32118992],
       [ 6.80551398],
       [ 7.28983805],
       [ 7.77416211],
       [ 8.25848618],
       [ 8.74281024],
       [ 9.22713431],
       [ 9.71145837],
       [10.19578244],
       [10.68010651],
       [11.16443057],
       [11.64875464],
       [12.1330787 ],
       [12.61740277],
       [13.10172683],
```

```
[13.5860509 ],
[14.07037496],
[14.55469903],
[15.03902309],
[15.52334716],
[16.00767122],
[16.49199529],
[16.97631936],
[17.46064342],
[17.94496749],
[18.42929155],
[18.91361562],
[19.39793968],
[19.88226375],
[20.36658781],
[20.85091188]])
```

마지막으로 , 원시 테이터를 먼저 플로팅 한 후 이 모델 적합을 플로팅 하여 시각화 가능

[14]:
```
plt.scatter(x, y)
plt.plot(xfit, yfit);
```



1.1.4 지도학습 예제: 붓꽃 분류

앞에서 나왔던 붓꽃 데이터세트를 사용해 일부 훈련 모델이 주어졌을때 나머지 레이블을 얼마나 잘 예측하는지 알아보자.

가우스 나이브 베이즈로 알려진 매우 간단한 생성 모델을 사용한다. 이 모델은 각 클래스가 가우스 분포로 정렬된 축으로 부터 비롯된다고 가정한다. 이 모델은 처리 속도가 빠르고 초모수를 선택할 필요가 없기때문에 정교한 모델을 통해 개선의 여지가 있는지 살펴보기 전에 기본 분류로 사용하기 좋다.

분류자료와 테스트 자료로 나누기 위해서 `train_test_split` 사용한다.

[15]:
```
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X_iris, y_iris,
                                                random_state=1)
```

데이터가 정리되면 위작업 절차에 따라 레이블을 예측 할 수 있다.:

[16]:
```
from sklearn.naive_bayes import GaussianNB # 1. choose model class
model = GaussianNB()                       # 2. instantiate model
model.fit(Xtrain, ytrain)                  # 3. fit model to data
y_model = model.predict(Xtest)             # 4. predict on new data
y_model
```

[16]:
```
array(['setosa', 'versicolor', 'versicolor', 'setosa', 'virginica',
       'versicolor', 'virginica', 'setosa', 'setosa', 'virginica',
       'versicolor', 'setosa', 'virginica', 'versicolor', 'versicolor',
       'setosa', 'versicolor', 'versicolor', 'setosa', 'setosa',
       'versicolor', 'versicolor', 'virginica', 'setosa', 'virginica',
       'versicolor', 'setosa', 'setosa', 'versicolor', 'virginica',
       'versicolor', 'virginica', 'versicolor', 'virginica', 'virginica',
       'setosa', 'versicolor', 'setosa'], dtype='<U10')
```

마지막으로 `accuracy_score`유틸리티를 사용해 예측한 레이블 중 실제 값과 일치하는 비율이 얼마나 되는지 확인 할 수 있다.:

[17]:
```
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
```

[17]:
```
0.9736842105263158
```

정확도가 97%넘었으니 이 매우 단순한 분류 알고리즘조차도 이 특징 데이터세트에 대해서는 효과적임을 알 수 있다.

1.1.5 비지도 학습 예제 : 붓꽃 차원

비지도 학습 문제의 예로 붓꽃 데이터를 좀 더 쉽게 시각화 할 수 있도록 차원을 축소하는 것을 살펴보자. 앞에서 본 붗꽃 데이터는 4차원 이었다. 즉, 각 표본에 대해서 특징 4가지 기록되어 있다.

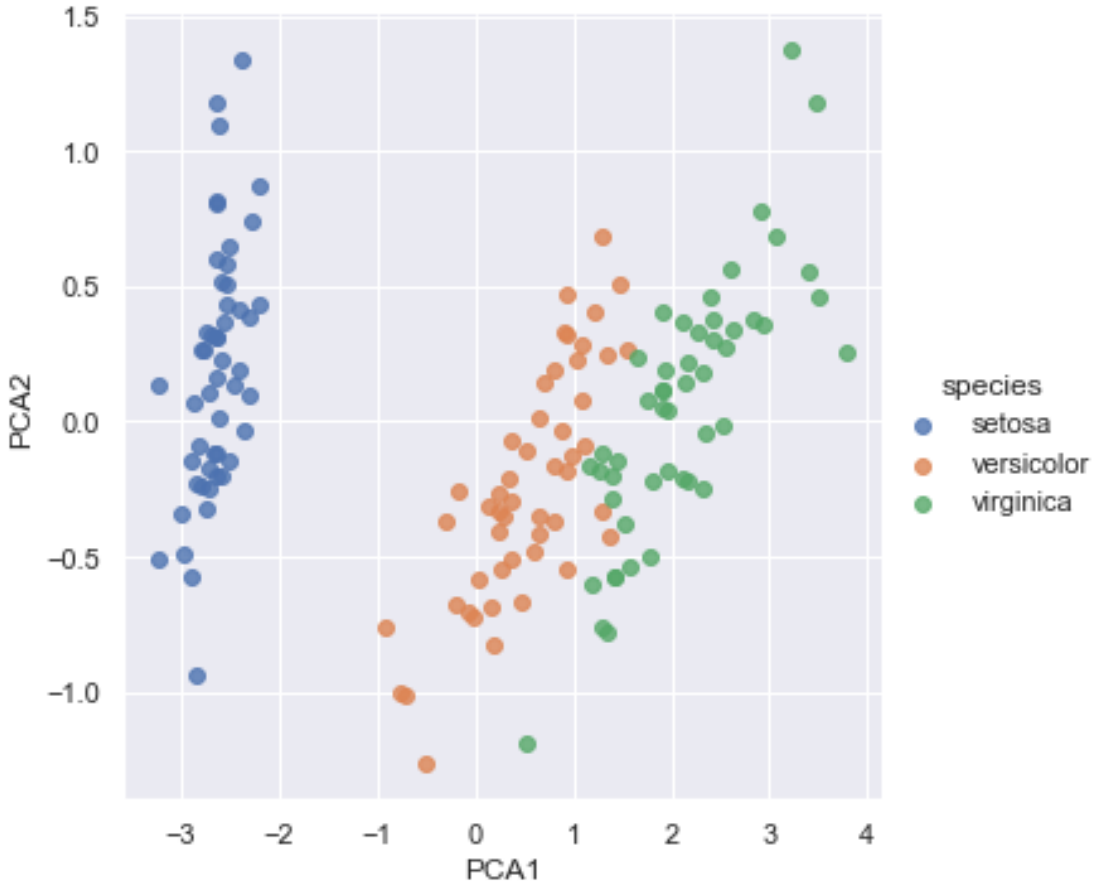
차원 축소 작업은 데이터의 근본적인 특징은 유지하면서 더 낮은 차원을 가지는 적절한 표현 방식이 존재하는지 알아보는 것이다. 종종 차원 축소가 데이터 시각화를 지원하기 위해 사용된다. 어쨋든 4차원이나 그 이상의 차원보다 2차원 데이터를 플로팅하는 것이 훨씬 더 쉽기 때문이다.

빠른 선형 차원 축소 기법인 주성분- 책 491페이지를 사용 하였다. 이 모델에 두개 성분 즉 그 데이터의 2차원 표현을 반환하도록 요청할 것이다.

[18]:
```
from sklearn.decomposition import PCA  # 1. 모델 클래스 선택
model = PCA(n_components=2)            # 2. 초모수로 모델 인스턴스화
model.fit(X_iris)                      # 3. Fit to data. Notice y is not␣
 ↪specified!
X_2D = model.transform(X_iris)         # 4. Transform the data to two dimensions
```

이제 결과를 플로팅 해보자 결과를 원래 붓꽃 DataFrame에 삽입하고 Seaborn's lmplot 를 사용하면 빠르게 할 수 있다.:

[19]:
```
iris['PCA1'] = X_2D[:, 0]
iris['PCA2'] = X_2D[:, 1]
sns.lmplot(x="PCA1", y="PCA2", hue='species', data=iris, fit_reg=False);
```



PCA 알고리즘이 붓꽃종 레이블에 대한 지식이 없는데도 2차원 표현에서 종이 매우 잘 분리돼 있음을 볼수 있다. 이것은 앞에 본 것처럼 비교적 단순 분류기법이 이 데이터세트에 효과적일 수도 있음을 의미한다.

1.1.6 비지도 학습: 붓꽃 군집화

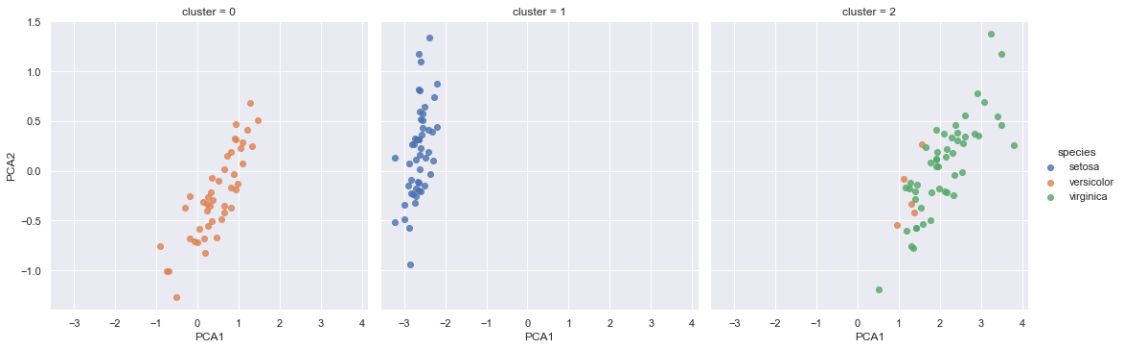
이번에는 군집화 기법을 붓꽃 데이터에 적용해 보자. 군집화 알고리즘은 아무 레이블도 참조하지 않고 데이터의 개별 그룹을 구한다. 책 p537에 자세하게 가우스 혼합모델 Gaussian mixture model (GMM)이 설명되어 있다. GMM은 데이터를 가우스 블롭의 컬렉션으로 모델링 하려고 한다.

다음과 같이 가우스 혼합모델을 적합 시킬수 있다.

[20]:
```
from sklearn.mixture import GaussianMixture      # 1. Choose the model class
#sklearn.mixture.GaussianMixture¶
model = GaussianMixture(n_components=3,
            covariance_type='full')  # 2. hyperparameters 모델 인스턴스화
model.fit(X_iris)                    # 3. 데이터에 적합  y는 지정하지 않음!
y_gmm = model.predict(X_iris)        # 4. 군집 레이블 결정
```

붓꽃 DataFrame에 군집 레이블을 추가하고 Seaborn사용해서 결과를 플로팅 할 것이다.:

[21]:
```
iris['cluster'] = y_gmm
sns.lmplot(x="PCA1", y="PCA2", data=iris, hue='species',
           col='cluster', fit_reg=False);
```



데이터를 군집 번호로 분할함으로써 GMM 알고리즘이 기존 레이블을 얼마나 잘 복구 했는지 정확하게 확인할 수 있다. *setosa* 종은 군집 0에 완벽하게 분리됐지만, *versicolor* 와 *virginica* 종은 서로 약간 섞여 있다. 이것은 곧 꽃마다 어느 종에 속하는지 알려주는 전문가가 없더라도 이 꽃들에 대한 측정값이 충분히 구별되어 간단한 군집 알고리즘 사용으로 서로 다른 종을 자동으로 식별 할 수 있다는 뜻이다. 이러한 종류의 알고리즘은 해당 분야의 전문가에게 관찰 중인 표본 간의 관계에 대한 단서를 제공 할 수도 있다.

1.1.7 응용 손으로 쓴 숫자 탐색

광학 문자 인식 문제 하나인 손으로 쓴 숫자를 식별하는 문제를 생각해보자. 이 문제는 결국 이미지에서 문자를 찾고 식별하는 것이다. 쉬운 방법을 택해 Scikit-Learn 라이브러리에 내장된 미리 포맷이 구성된 숫자 집합을 사용한다.

1.1.8 숫자 데이터 적재 및 시각화

Scikit-Learn데이터 접근 인터페이스를 사용해 이 데이터를 살펴보자:

[22]:
```
from sklearn.datasets import load_digits
digits = load_digits()
digits.images.shape
```

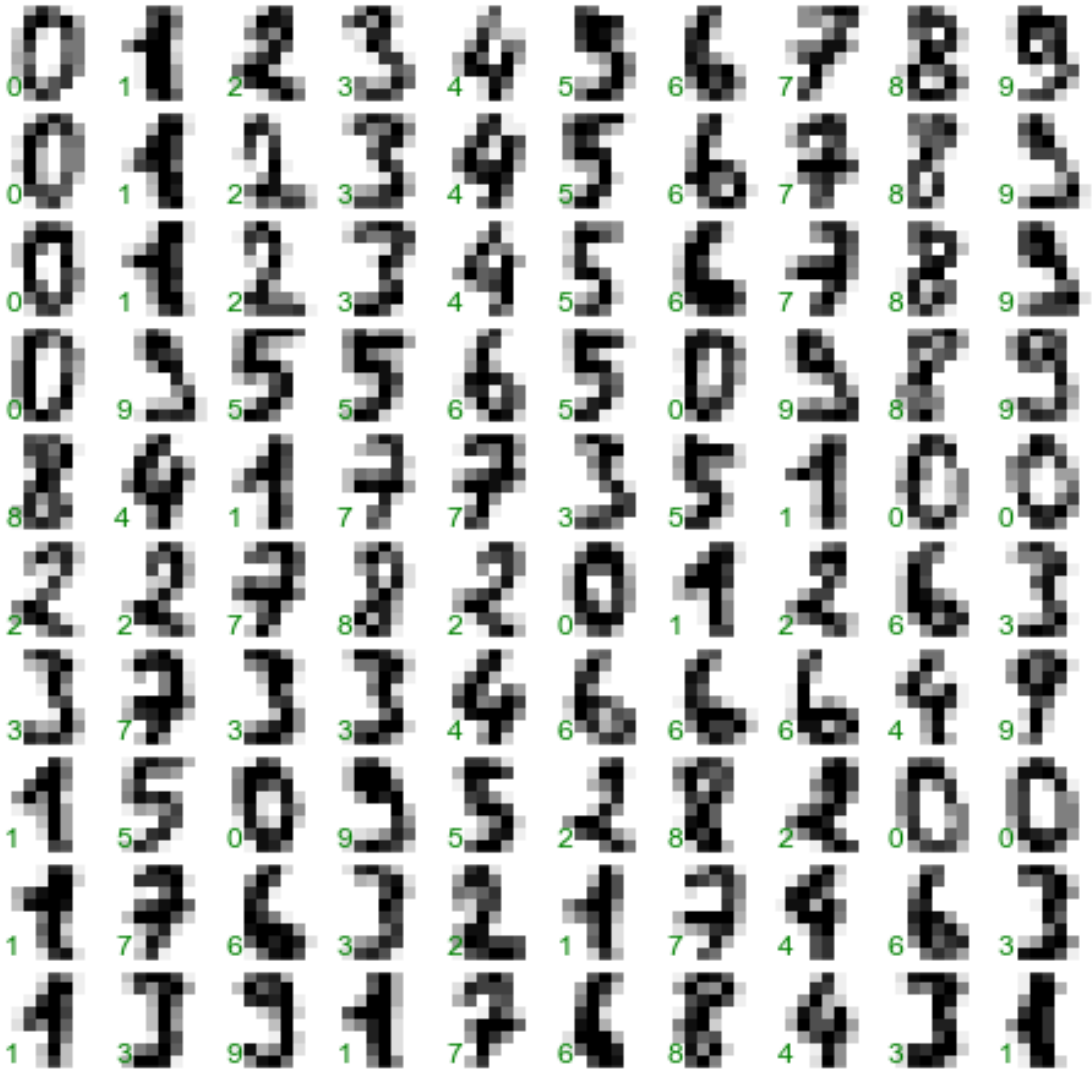
[22]:
```
(1797, 8, 8)
```

이미지 데이터는 3차원 배열로 1797개의 표본을 가지고 있으며, 각 표본은 8x8 픽셀 그리드로 구성되어 있다. 처음 100개를 시각화해보자:

[23]:
```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))

for i, ax in enumerate(axes.flat):
    ax.imshow(digits.images[i], cmap='binary', interpolation='nearest')
    ax.text(0.05, 0.05, str(digits.target[i]),
            transform=ax.transAxes, color='green')
```



이 데이터를 Scikit-Learn에서 작업하면 2차원 [n_samples, n_features]표현이 필요하다. 이미지의 각 픽셀을 특징으로 취급해 이 작업을 할 수 있다. 즉, 픽셀 배열을 평평하게 펴서 각 숫자를 나타내는 픽셀값을 길이 64배열로 바꾸면 된다. 이밖에도 각 숫자에 대해 미리 결정된 레이블을 제공하는 대상 배열이 필요하다. 이 두량은 숫자 데이터세트의 data 와 target속성 아래 구축되어 있다.:

[24]:
```
X = digits.data
X.shape
```

[24]:
```
(1797, 64)
```

[25]:
```
y = digits.target
y.shape
```

[25]:
```
(1797,)
```

1,797 개의 표본과 64 개의 특징이 있음을 알수 있다.

1.1.9 비지도 학습 : 차원 축소

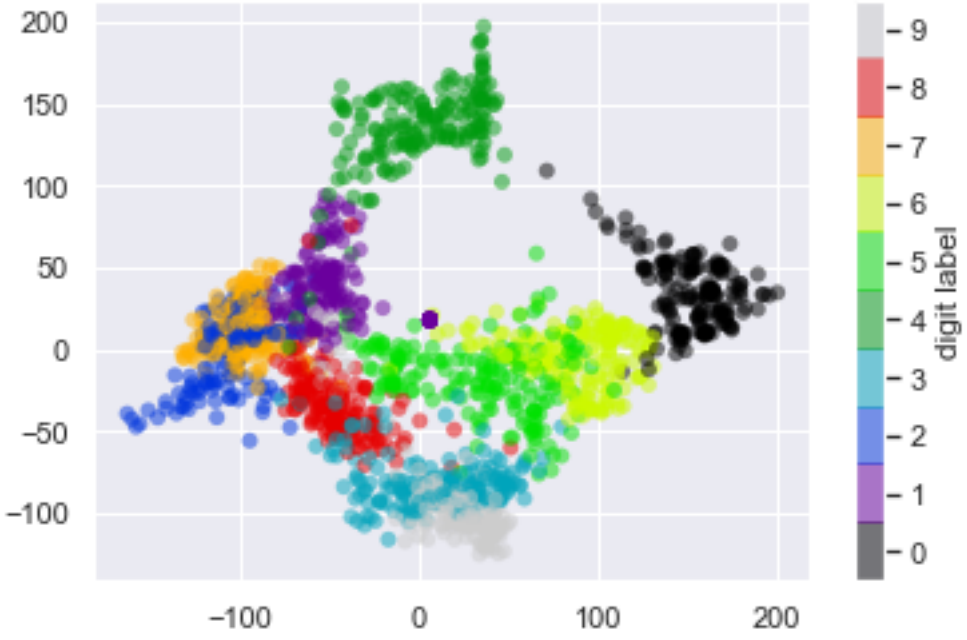
64차원 모수 공간에 점을 시각화하고 싶어도 높은 차원의 공간에 점을 효과적으로 시각화하기 어렵다. 대신 비지도학습 방식을 사용해 차원을 2차원으로 줄일 수 있다. 등거리사상 *Isomap* (504쪽)이라는 다양체 학습 알고리즘을 사용해 2차원으로 변환하고자 한다.

[26]:
```
from sklearn.manifold import Isomap
iso = Isomap(n_components=2)
iso.fit(digits.data)
data_projected = iso.transform(digits.data)
data_projected.shape
```

[26]:
```
(1797, 2)
```

사영된 데이터가 이제2차원이 되었다. 그렇다면 이 데이터를 플로팅해 그 구조부터 학습할 내용이 있는지 확인해보고자 한다.:

[27]:
```
plt.scatter(data_projected[:, 0], data_projected[:, 1], c=digits.target,
            edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('nipy_spectral', 10))
plt.colorbar(label='digit label', ticks=range(10))
plt.clim(-0.5, 9.5);
```



이 플롯은 다양한 숫자가 더큰 64차원 공간에서 얼마나 잘 구분되는지에 대해 직관을 제공한다. 예를들어 0(검은색) 1(보라색)은 모수공간에서 거의 겹치지 않는다. 직관적으로 충분히 이해된다. 0은 이미지의

가운데가 비어있지만 1은 일반적으로 가운데에 잉크가 묻는다. 반면 1과 4는 거의 연속인 스펙트럼을 가진 것처럼 보인다. 1을 쓸때 위에 모자를 다는 사람이 있는데 이 경우 1과 4가 비슷해 보일 수 있다.

전반적으로 모수 공간에서 각 그룹이 매우잘 구분되는 편이다. 이로써 매우 단순한 지도 분류알고리즘도 이 데이터에 적절하게 적용될 수 있다.

1.1.10 숫자 분류

분류 알고리즘을 숫자에 적용하자. 앞에서 나온 붓꽃 데이터와 마찬가지로 이 데이터를 훈련자로와 테스트 자료로 나누고 가우스 나이브 베이즈 모델을 적합시킬 것이다.:

[28]:
```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=0)
```

[29]:
```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(Xtrain, ytrain)
y_model = model.predict(Xtest)
```

모델을 예측 했으니 이제 테스트 자료의 실제 값을 예측값과 비교해 모델의 정확도를 측정할 수 있다:

[30]:
```
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
```

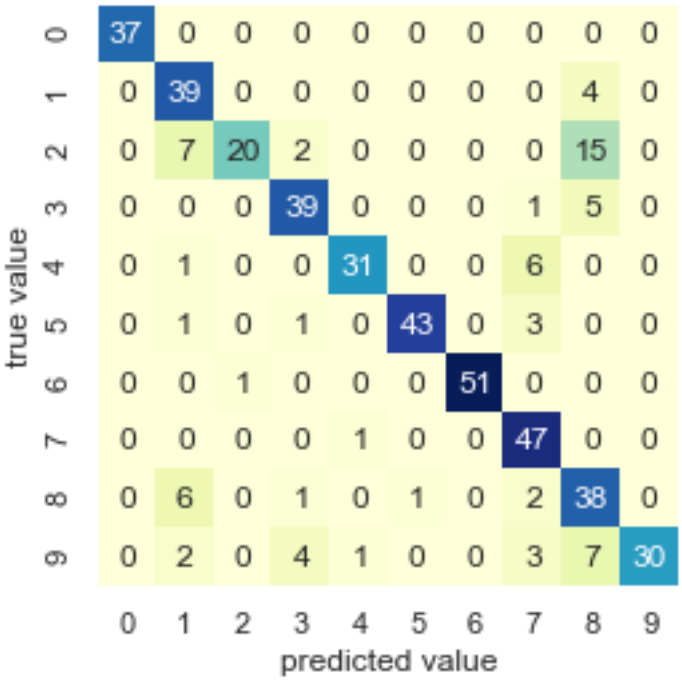
[30]:
```
0.8333333333333334
```

이렇게 간단한 모델로 숫자의 분류의 정확도가 약 80%가 된다. 그러나 이 숫자 하나로는 어디에서 제대로 예측을 못한 것인지 알 수 없다. 이를 알아내는 한가자 좋은 방법은 오차행렬 (*confusion matrix*)을 사용하는 것이다. 여기서는 Scikit-Learn으로 계산하고 Seaborn으로 플로팅할것이다:

[31]:
```
from sklearn.metrics import confusion_matrix

mat = confusion_matrix(ytest, y_model)

sns.heatmap(mat, square=True, annot=True, cbar=False, cmap="YlGnBu")
plt.xlabel('predicted value')
plt.ylabel('true value');
```

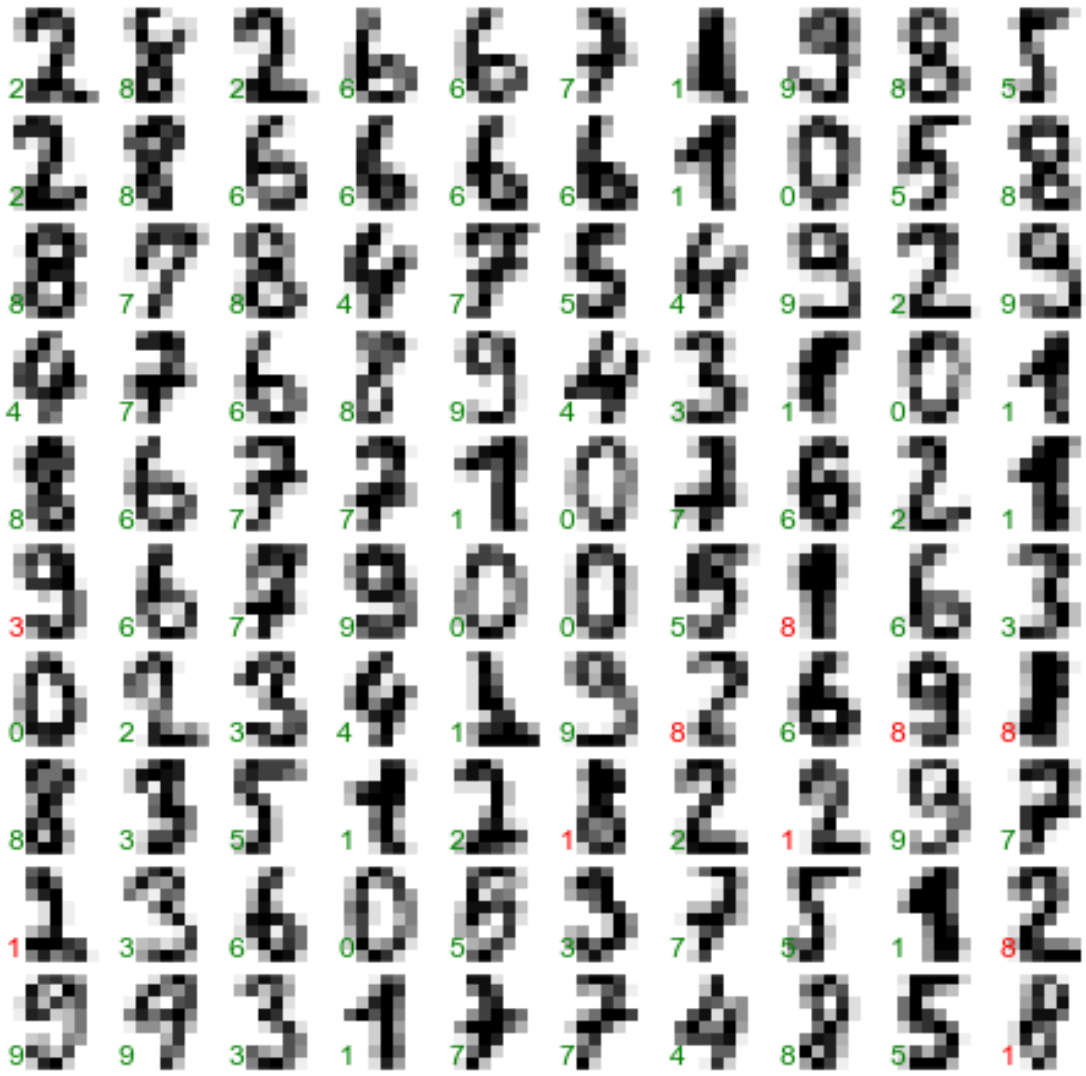


이 그림을 통해 레이블이 잘못 분류된 점들이 주로 어디에 있는지 알 수 있다. 예를 들면 많은 2가 1이나 8로 잘못 분류됐다. 이 모델의 특성에 대한 직관을 얻는 또 다른 방법은 모델이 예측한 레이블로 입력 데이터를 다시 플로팅 하는 것이다. 예제에서는 올바른 레이블은 녹색 잘못된 레이블은 빨간색을 사용할 것이다. :

[32]:
```
fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))

test_images = Xtest.reshape(-1, 8, 8)

for i, ax in enumerate(axes.flat):
    ax.imshow(test_images[i], cmap='binary', interpolation='nearest')
    ax.text(0.05, 0.05, str(y_model[i]),
            transform=ax.transAxes,
            color='green' if (ytest[i] == y_model[i]) else 'red')
```



이 데이터의 하위 집합을 조사해보면 알고리즘이 적절하게 수행되지 않는 곳이 어디인지에 대한 통찰력을 얻을 수 있다. 80%라는 분류율을 넘어서기 위해 서포트벡터 머신(p462)이나 랜덤포레스트(P478) 그 밖의 다른 분류방식 등이 더 정교한 알고리즘을 이용할 수도 있다.

[ ]: