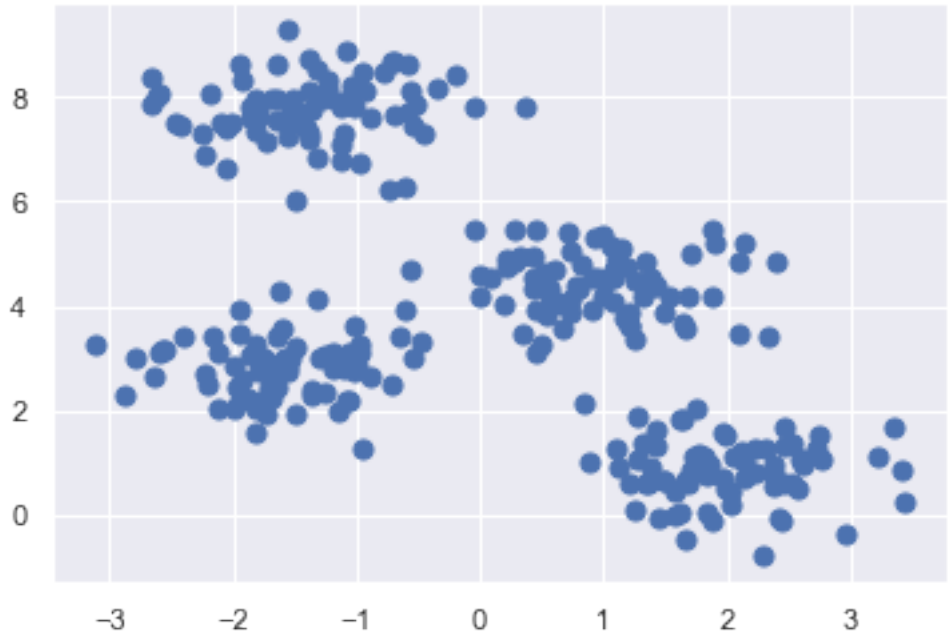
??_k-means??

July 8, 2019

[1]:
```
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
```

[2]:
```
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.6,␣
 ↪random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50)
```

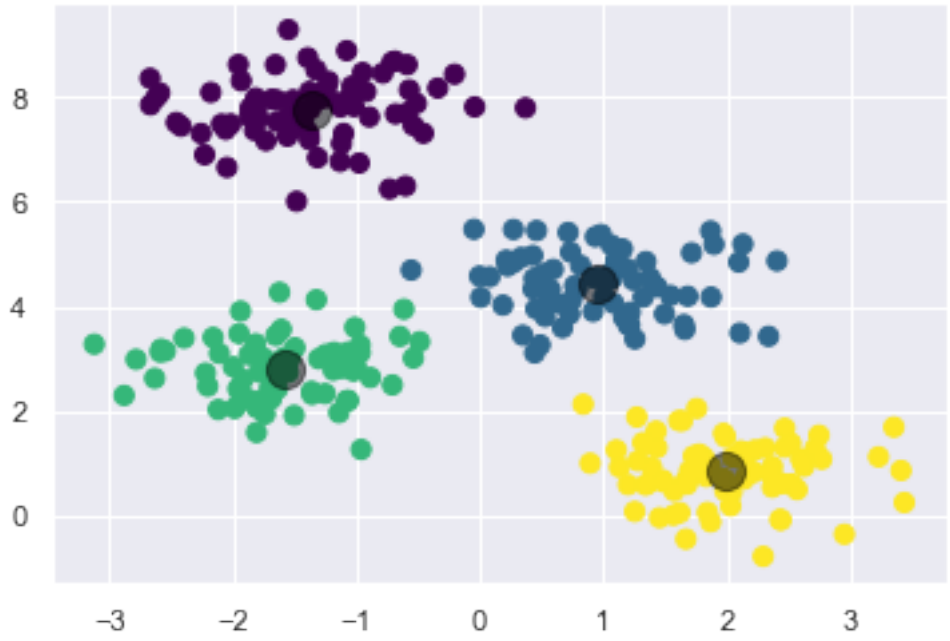
[2]:
```
<matplotlib.collections.PathCollection at 0x11fa7ec50>
```



[7]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
```

[7]: <matplotlib.collections.PathCollection at 0x122859dd8>

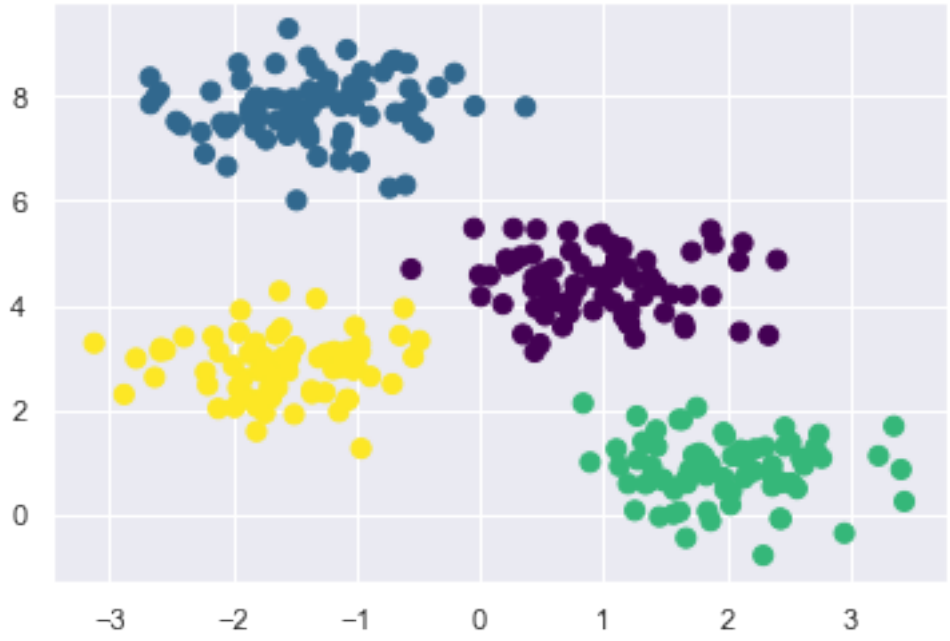


1 k-means

[10]:
```
#a:
#b:
from sklearn.metrics import pairwise_distances_argmin

def find_clusters(X, n_clusters, rseed=2):
    #1.
    rng = np.random.RandomState(rseed)
    i = rng.permutation(X.shape[0])[: n_clusters]
    centers = X[i]

    while True:
        #2a:
        labels = pairwise_distances_argmin(X, centers)

        #2b:
        new_centers = np.array([X[labels == i].mean(0)
```

```
                                for i in range(n_clusters)])

        #2c:
        if np.all(centers == new_centers):
            break
        centers = new_centers

    return centers, labels

centers, labels = find_clusters(X, 4)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap="viridis")
```

[10]: <matplotlib.collections.PathCollection at 0x1229d49b0>

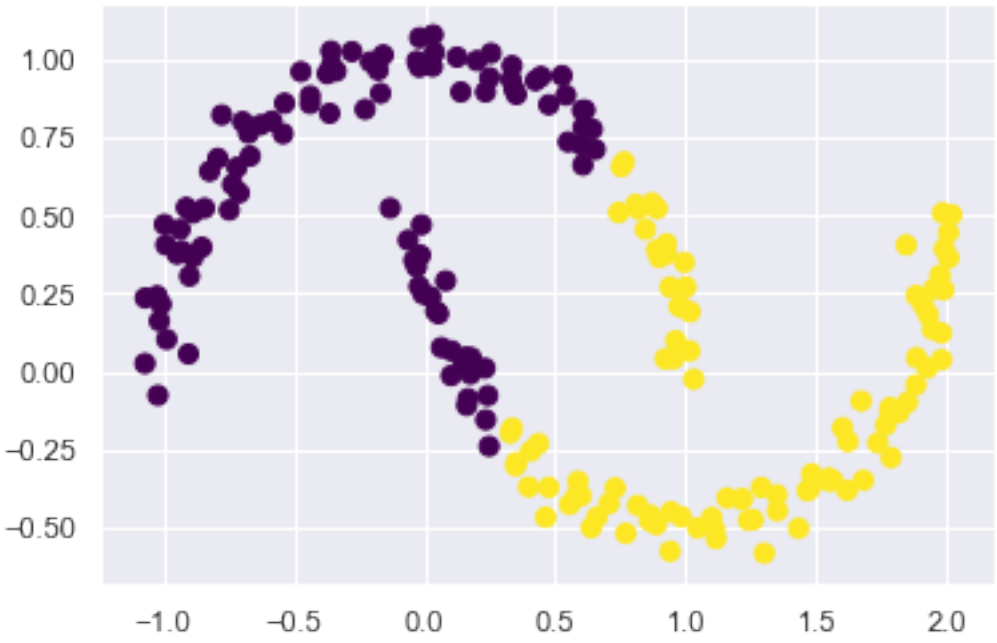


2 k-means

[12]:
```
from sklearn.datasets import make_moons
X, y = make_moons(200, noise=0.05, random_state=0)
```

[15]:
```
#k-means
labels = KMeans(2, random_state=0).fit_predict(X)
plt.scatter(X[:, 0], X[: ,1], c=labels, s=50, cmap='viridis')
```

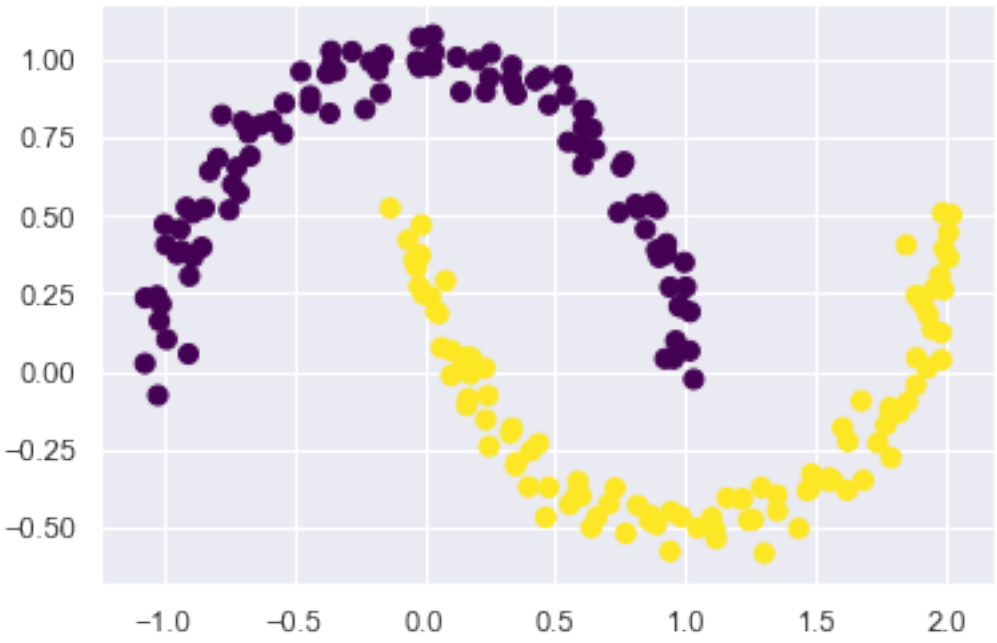
[15]: <matplotlib.collections.PathCollection at 0x1226c0048>



[13]:
```
#k-means
#k-meansSpectralClustering
from sklearn.cluster import SpectralClustering
model = SpectralClustering(n_clusters=2, affinity='nearest_neighbors',␣
 ↪assign_labels='kmeans')
labels = model.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
```

```
/usr/local/miniconda3/lib/python3.7/site-
packages/sklearn/manifold/spectral_embedding_.py:235: UserWarning: Graph is not
fully connected, spectral embedding may not work as expected.
  warnings.warn("Graph is not fully connected, spectral embedding"
```

[13]:
```
<matplotlib.collections.PathCollection at 0x122a3cd30>
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

??_?????

July 8, 2019

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
```

[2]:
```
#
#200
rng = np.random.RandomState(1)
X = np.dot(rng.rand(2, 2), rng.randn(2, 200)).T
plt.scatter(X[:, 0], X[:, 1])
plt.axis('equal')
```

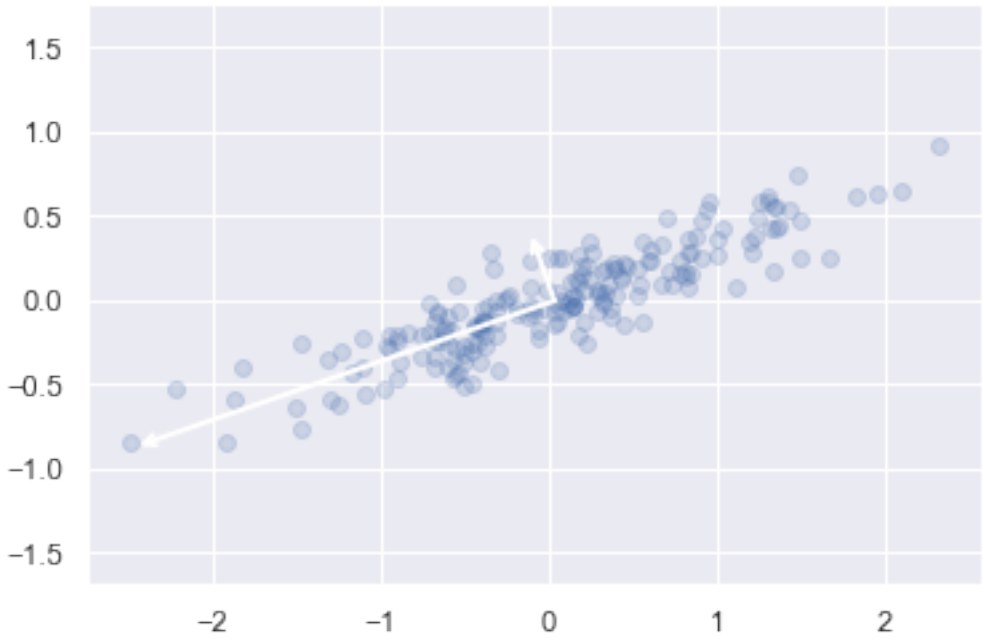
[2]:
```
(-2.7391278364515688,
 2.5801310701596343,
 -0.9477947579593762,
 1.019590430670684)
```

[3]:
```
#xyxy
#
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X)
#
print(pca.components_)
print(pca.explained_variance_)
```

```
[[-0.94446029 -0.32862557]
 [-0.32862557  0.94446029]]
[0.7625315 0.0184779]
```

[11]:
```
#""
def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops = dict(arrowstyle='->',
                      linewidth=2,
                      shrinkA=0,
                      shrinkB=0)
    ax.annotate('', v1, v0, arrowprops=arrowprops)

#
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
for length, vector in zip(pca.explained_variance_, pca.components_):
    v = vector * 3 * np.sqrt(length)
    draw_vector(pca.mean_, pca.mean_ + v)
plt.axis('equal')
```

[11]:
```
(-2.7391278364515688,
 2.5801310701596343,
 -0.9477947579593762,
 1.019590430670684)
```

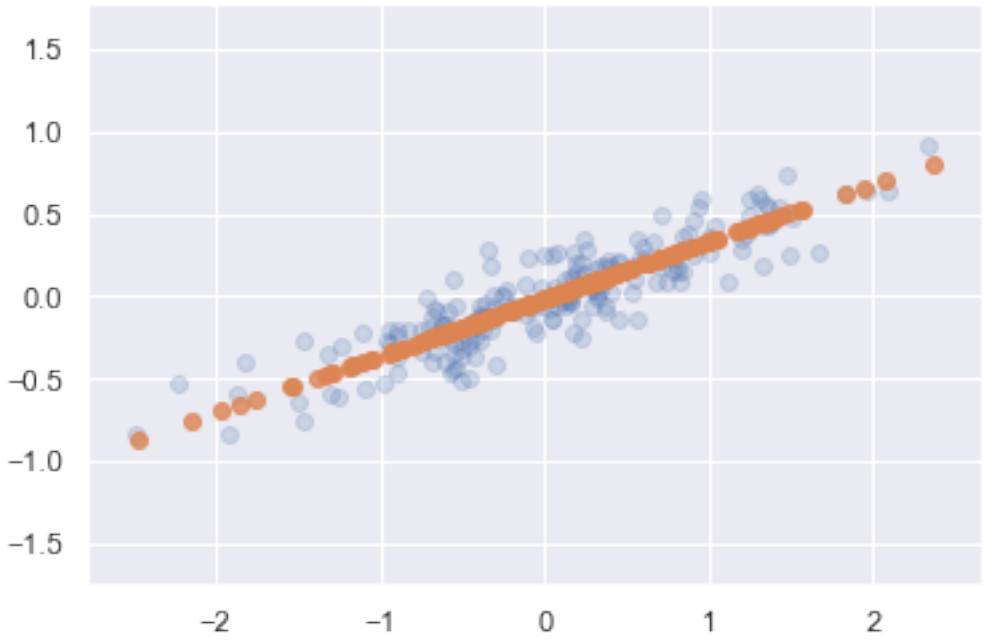


[12]:
```
#PCA
#PCA:
pca = PCA(n_components=1)
pca.fit(X)
X_pca = pca.transform(X)
print("original shape: ", X.shape)
print("transformed shape: ", X_pca.shape)
```

```
original shape:  (200, 2)
transformed shape:  (200, 1)
```

[17]:
```
#
X_new = pca.inverse_transform(X_pca)#()()
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
plt.scatter(X_new[:, 0], X_new[:, 1], alpha=0.8)
plt.axis('equal')
```

[17]:
```
(-2.771528780690219, 2.661757596590678, -0.9964674432667128, 1.021908177590081)
```


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2

??_????????

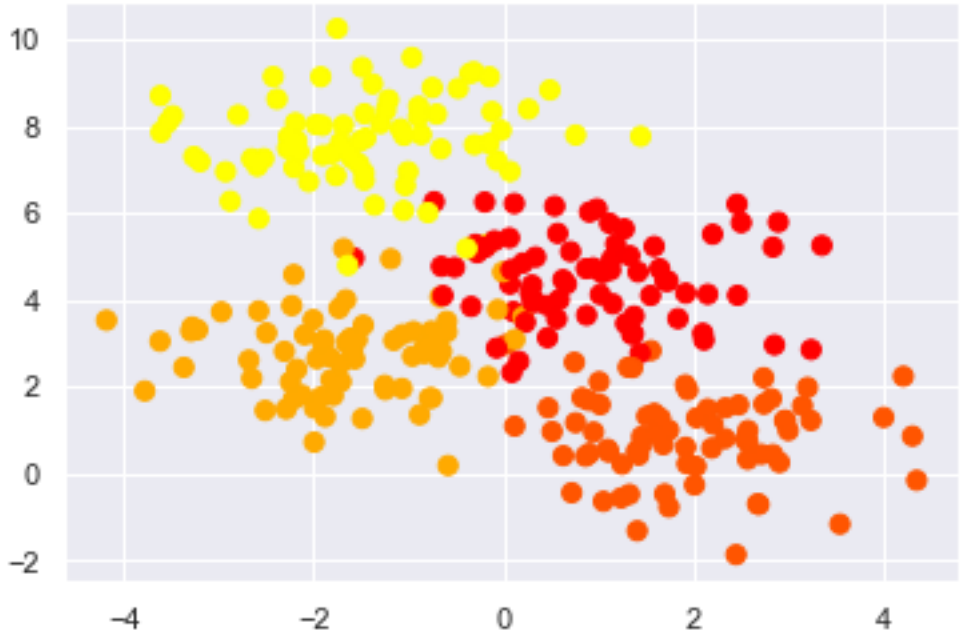
July 8, 2019

1

[30]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
```

[31]:
```
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=300, centers=4, random_state=0, cluster_std=1.0)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
```

[31]:
```
<matplotlib.collections.PathCollection at 0x120306a90>
```

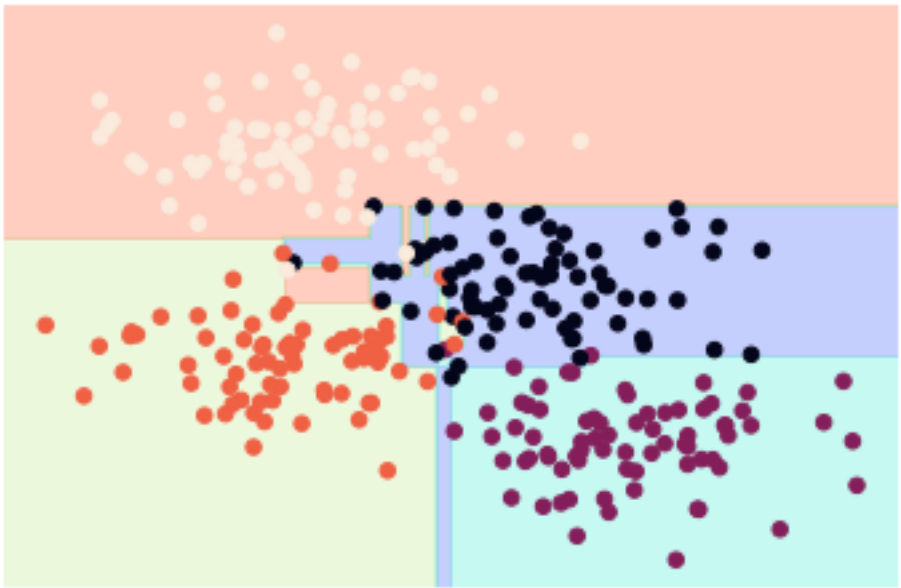


[32]:
```
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
tree = DecisionTreeClassifier().fit(X, y)
```

[73]:
```
#
def visualize_classifier(model, X, y, ax=None, cmap='rainbow'):
    ax = ax or plt.gca()

    #
    plt.scatter(X[:, 0], X[:, 1], c=y, s=30, clim=(y.min(), y.max()), zorder=3)

    ax.axis('tight')
    ax.axis('off')
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    #
    model.fit(X, y)
    xx, yy = np.meshgrid(np.linspace(*xlim, num=200),
                         np.linspace(*ylim, num=200))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

    #
    n_classes = len(np.unique(y))
    contours = ax.contourf(xx, yy, Z, alpha=0.3,
                           levels=np.arange(n_classes + 1) - 0.3,
                           cmap=cmap, clim=(y.min(), y.max()), zorder=1)
    ax.set(xlim=xlim, ylim=ylim)
```

[74]:
```
visualize_classifier(DecisionTreeClassifier(), X, y)
```

```
/usr/local/miniconda3/lib/python3.7/site-packages/ipykernel_launcher.py:23:
UserWarning: The following kwargs were not used by contour: 'clim'
```



[76]:
```
import helpers_05_08
helpers_05_08.plot_tree_interactive(X, y)
```

```
interactive(children=(Dropdown(description='depth', index=1, options=(1, 5), value=5), Output()
```

[76]:
```
<function
helpers_05_08.plot_tree_interactive.<locals>.interactive_tree(depth=5)>
```

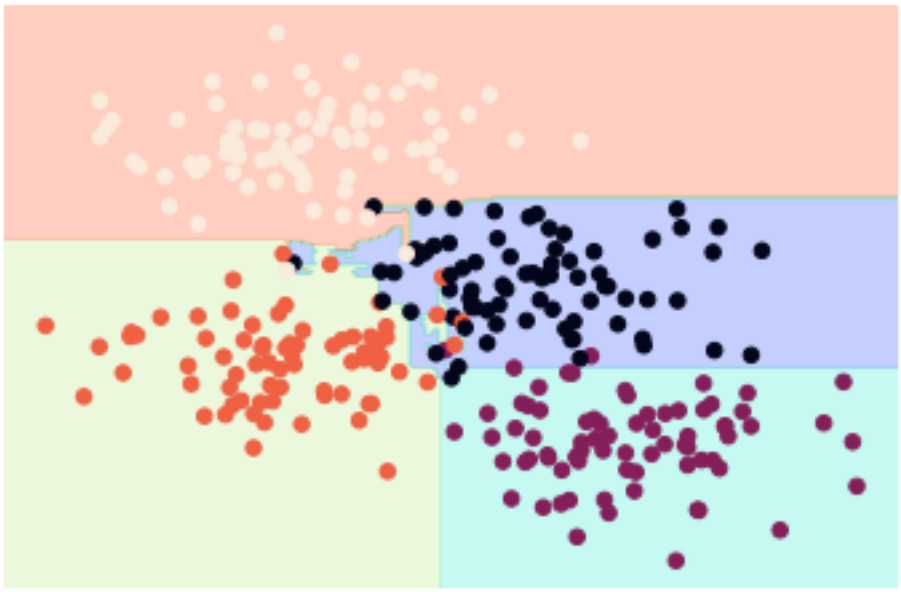
[77]:
```
#
import helpers_05_08
helpers_05_08.randomized_tree_interactive(X, y)
```

```
interactive(children=(Dropdown(description='random_state', options=(0, 100), value=0), Output()
```

2

[78]:
```
#:
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier

tree = DecisionTreeClassifier()
bag = BaggingClassifier(tree, n_estimators=100, max_samples=0.8, random_state=1)

visualize_classifier(bag, X, y)
```

```
/usr/local/miniconda3/lib/python3.7/site-packages/ipykernel_launcher.py:23:
UserWarning: The following kwargs were not used by contour: 'clim'
```



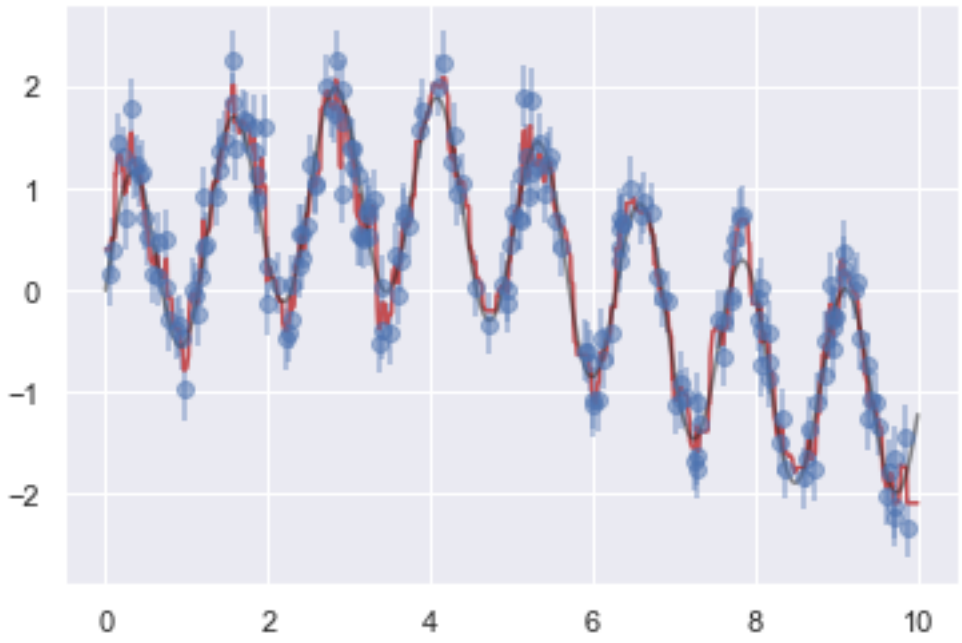
3

[84]:
```
rng = np.random.RandomState(42)
x = 10 * rng.rand(200)

def model(x, sigma=0.3):
    fast_oscillation = np.sin(5 * x)
    slow_oscillation = np.sin(0.5 * x)
    noise = sigma * rng.randn(len(x))

    return slow_oscillation + fast_oscillation + noise

y = model(x)
plt.errorbar(x, y, 0.3, fmt='o')
```

[84]:
```
<ErrorbarContainer object of 3 artists>
```

[86]:
```
#
from sklearn.ensemble import RandomForestRegressor
forest = RandomForestRegressor(200)
forest.fit(x[:, None], y)

xfit = np.linspace(0, 10, 1000)
yfit = forest.predict(xfit[:, None])
ytrue = model(xfit, sigma=0)

plt.errorbar(x, y, 0.3, fmt='o', alpha=0.5)
plt.plot(xfit, yfit, '-r')
plt.plot(xfit, ytrue, '-k', alpha=0.5)
```

[86]:
```
[<matplotlib.lines.Line2D at 0x12083bf98>]
```


2
1
0
−1
−2
0
2
4
6
8
10

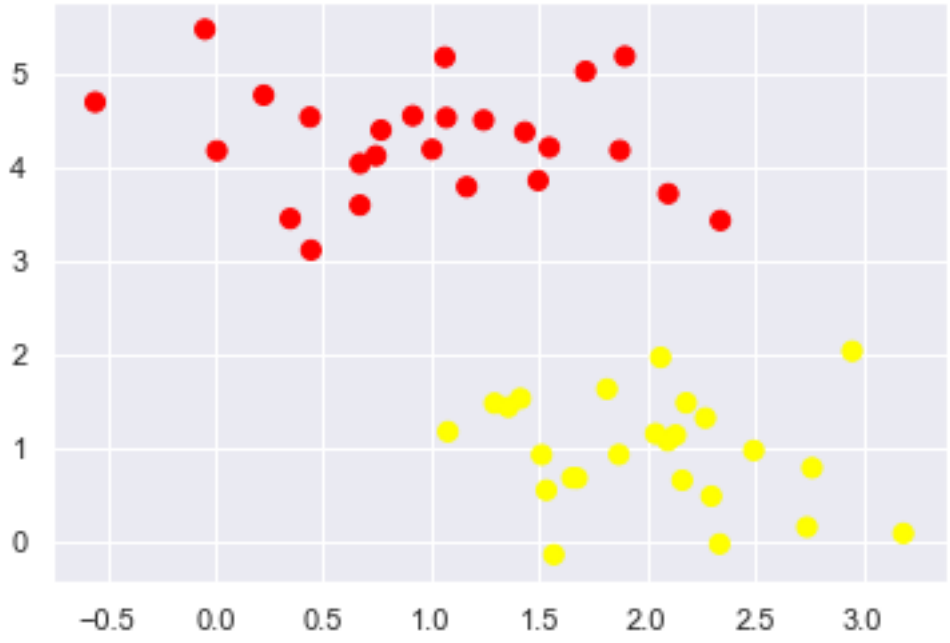
??_?????

July 8, 2019

[56]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

import seaborn as sns; sns.set()
```

[57]:
```
from sklearn.datasets.samples_generator import make_blobs
X, y = make_blobs(n_samples=50, centers=2,
                  random_state=0, cluster_std=0.60)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
```

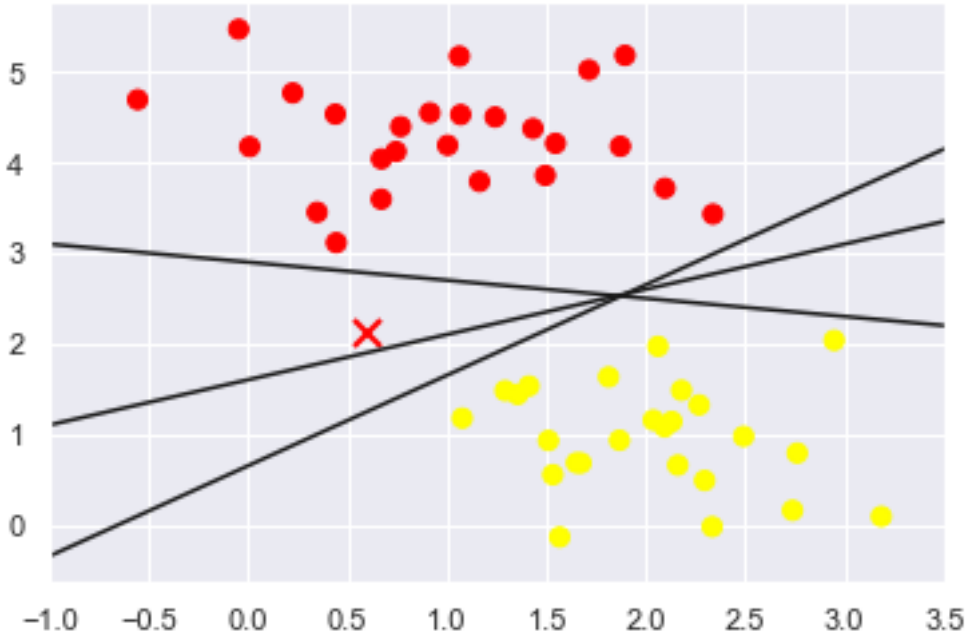
[57]:
```
<matplotlib.collections.PathCollection at 0x11f5fe828>
```



[58]:
```
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plt.plot([0.6], [2.1], 'x', color='red', markeredgewidth=2, markersize=10)
```

```
for m, b in [(1, 0.65), (0.5, 1.6), (-0.2, 2.9)]:
    plt.plot(xfit, m * xfit + b, '-k')

plt.xlim(-1, 3.5)
```

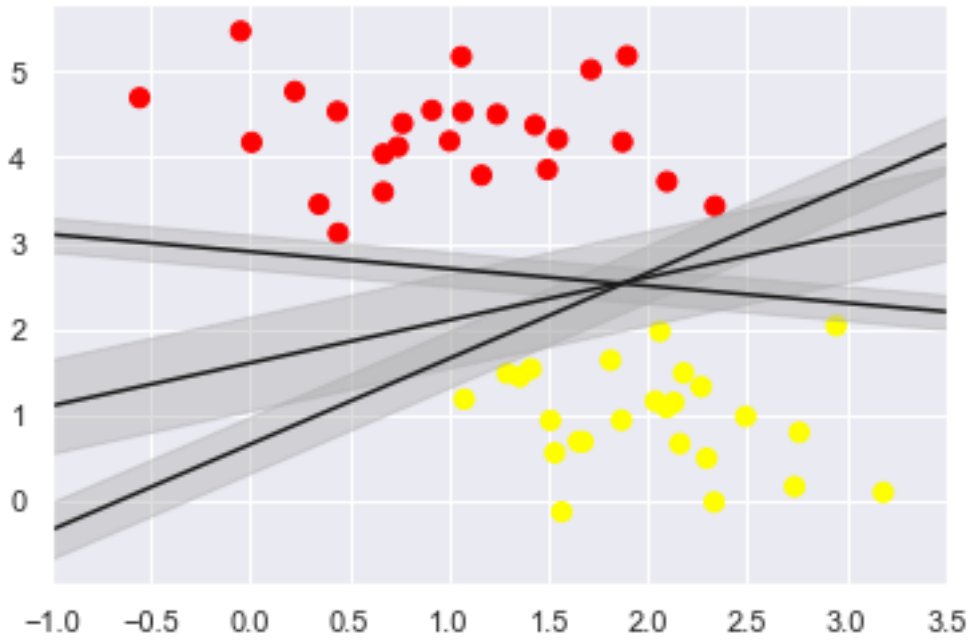
[58]: (-1, 3.5)



1

[59]:
```
#
#-
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
    yfit = m * xfit + b
    plt.plot(xfit, yfit, '-k')
    plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',␣
 ↪color='#AAAAAA', alpha=0.4)
plt.xlim(-1, 3.5)
```

[59]: (-1, 3.5)



2

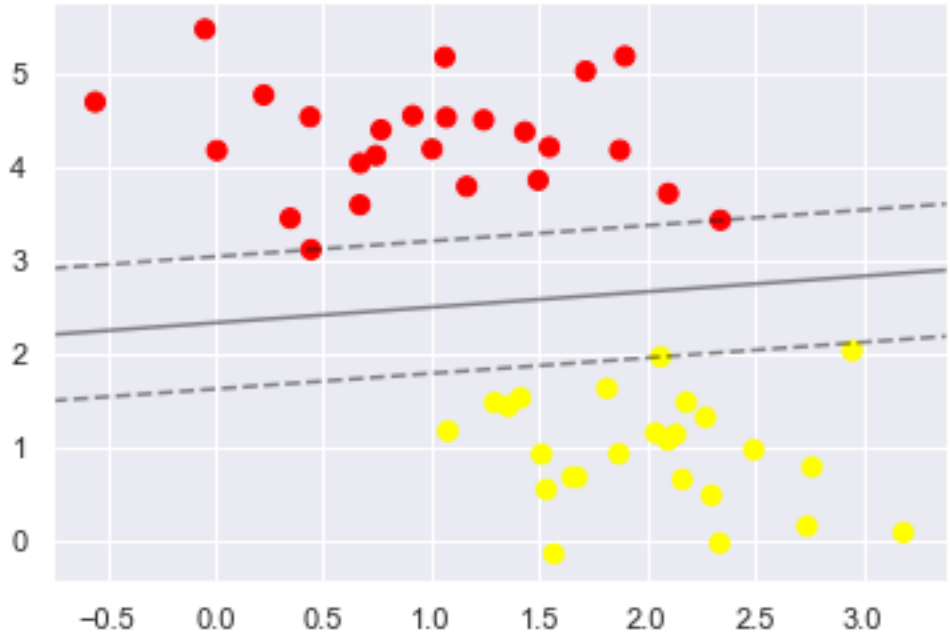
[60]:
```
from sklearn.svm import SVC
model = SVC(kernel='linear', C=1E10)
model.fit(X, y)
```

[60]:
```
SVC(C=10000000000.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

[61]:
```
#SVM
def plot_svc_decision_function(model, ax=None, plot_support=True):
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    #
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)
```

```
    #
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1],
               alpha=0.5,
               linestyles=['--', '-', '--'])

    #
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=200, linewidth=1, facecolor='none')
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
```

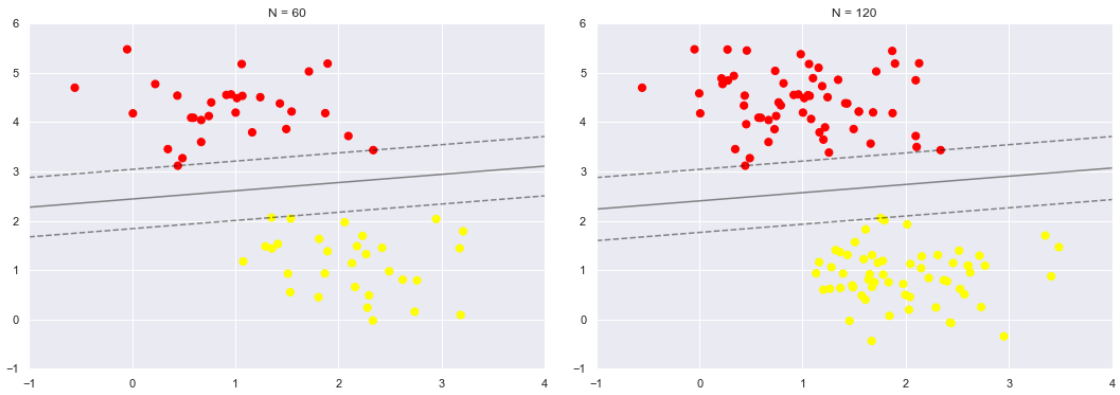
[62]:
```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(model)
```



[63]:
```
#
#sklearnsupport_vectors_
model.support_vectors_
```

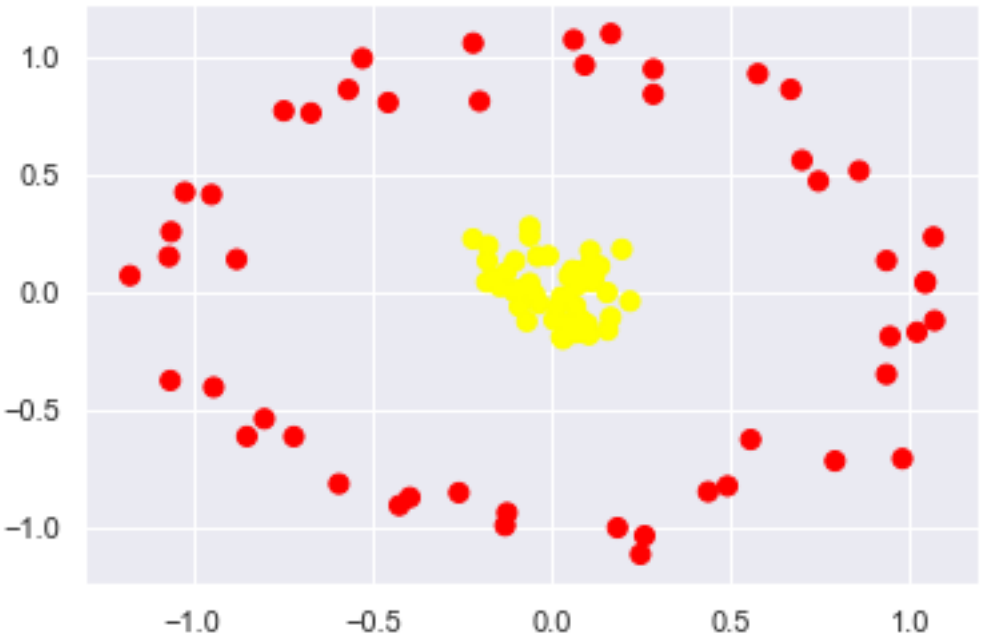
[63]:
```
array([[0.44359863, 3.11530945],
       [2.33812285, 3.43116792],
       [2.06156753, 1.96918596]])
```

[64]:
```
#60120
def plot_svm(N=10, ax=None):
    X, y = make_blobs(n_samples=N, centers=2,
                      random_state=0, cluster_std=0.6)
    X = X[:N]
    y = y[:N]
    model = SVC(kernel='linear', C=1E10)
    model.fit(X, y)

    ax = ax or plt.gca()
    ax.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
    ax.set_xlim(-1, 4)
    ax.set_ylim(-1, 6)
    plot_svc_decision_function(model, ax)

fig, ax = plt.subplots(1, 2, figsize=(16, 6))
fig.subplots_adjust(left=0.0625, right=0.95, wspace=0.1)
for axi, N in zip(ax, [60, 120]):
    plot_svm(N, axi)
    axi.set_title('N = {0}'.format(N))
```



[83]:
```
#SVM
#
from sklearn.datasets.samples_generator import make_circles
X, y = make_circles(100, factor=.1, noise=.1)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
```

[83]:
```
<matplotlib.collections.PathCollection at 0x11ff49be0>
```



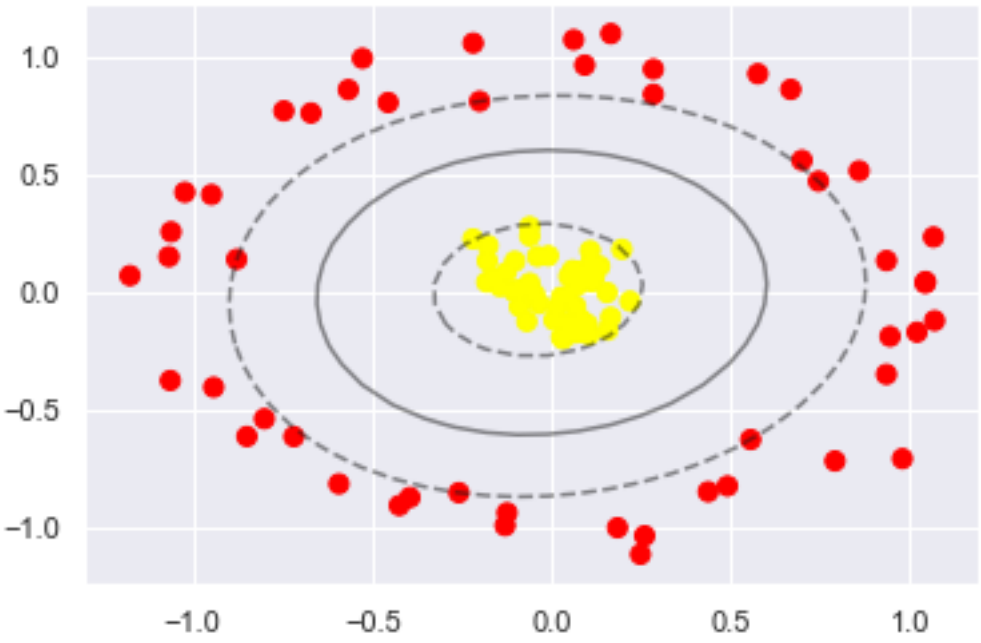
[84]:
```
#
#SVMRBF()
clf = SVC(kernel='rbf', C=1E6)
clf.fit(X, y)
```

```
/usr/local/miniconda3/lib/python3.7/site-packages/sklearn/svm/base.py:193:
FutureWarning: The default value of gamma will change from 'auto' to 'scale' in
version 0.22 to account better for unscaled features. Set gamma explicitly to
'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

[84]:
```
SVC(C=1000000.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

[85]:
```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(clf)
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=300,␣
 ↪lw=1, facecolors='none')
```

[85]:
```
<matplotlib.collections.PathCollection at 0x1022b8a90>
```


1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

[ ]:

??_???????

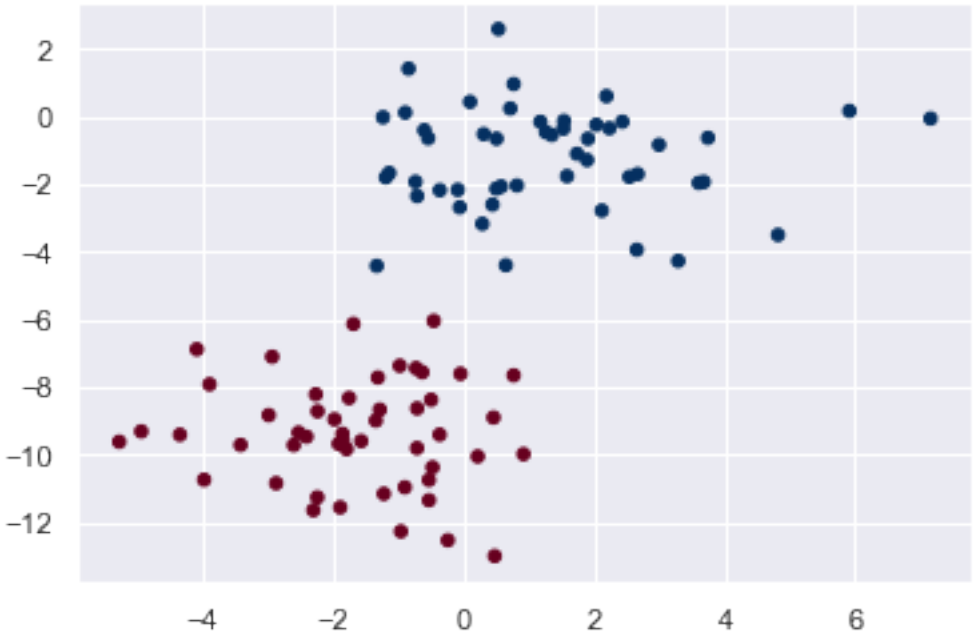
July 8, 2019

[1]:
```
#
#
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
```

1

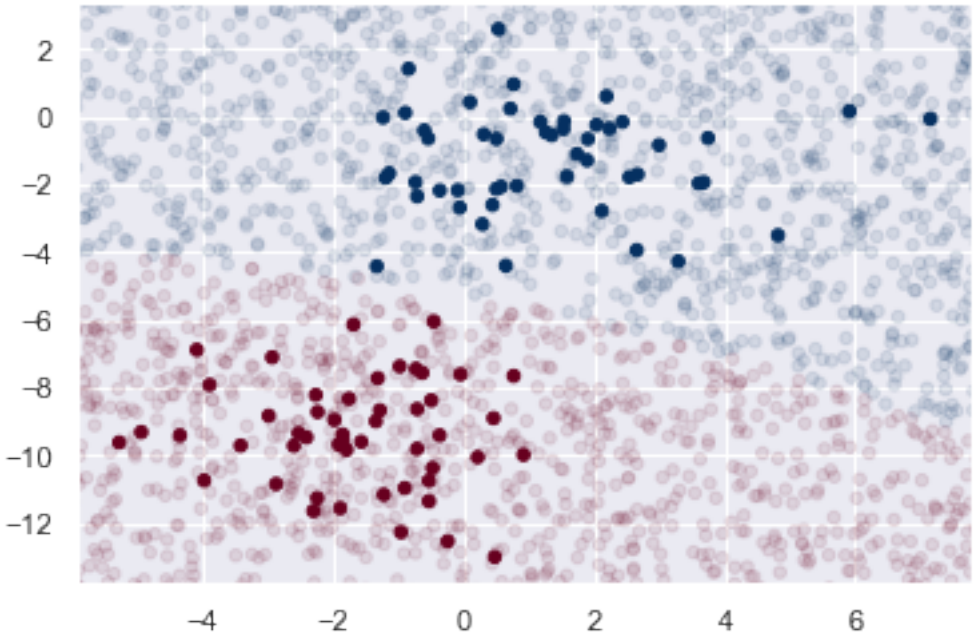
[34]:
```
from sklearn.datasets import make_blobs
X, y = make_blobs(100, 2, centers=2, random_state=2, cluster_std=1.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=20, cmap='RdBu')
```

[34]:
```
<matplotlib.collections.PathCollection at 0x1214a8080>
```



[33]:
```
#
#
#P(X|y),
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X, y)
#
rng = np.random.RandomState(0)
Xnew = [-6, -14] + [14, 18] * rng.rand(2000, 2)
ynew = model.predict(Xnew)
plt.scatter(X[:, 0], X[:, 1], c=y, s=20, cmap='RdBu')

lim = plt.axis()
plt.scatter(Xnew[:, 0], Xnew[:, 1], c=ynew, s=20, cmap='RdBu', alpha=0.1)
plt.axis(lim)
```

[33]:
```
(-5.897901072336819,
 7.784913423883647,
 -13.787255635247188,
 3.3747656397674346)
```



[35]:
```
#predict_proba
yprob = model.predict_proba(Xnew)
yprob[-8:].round(2)
```

[35]:
```
array([[0.89, 0.11],
       [1.  , 0.  ],
```

```
[1.  , 0.  ],
[1.  , 0.  ],
[1.  , 0.  ],
[1.  , 0.  ],
[0.  , 1.  ],
[0.15, 0.85]])
```

2

[40]:
```
#
from sklearn.datasets import fetch_20newsgroups
data = fetch_20newsgroups()
data.target_names
```

[40]:
```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

[77]:
```
#4
categories = ['talk.religion.misc', 'soc.religion.christian', 'sci.space',␣
 ↪'comp.graphics']
train = fetch_20newsgroups(subset='train', categories=categories)
test = fetch_20newsgroups(subset='test', categories=categories)
print(train.target_names)
```

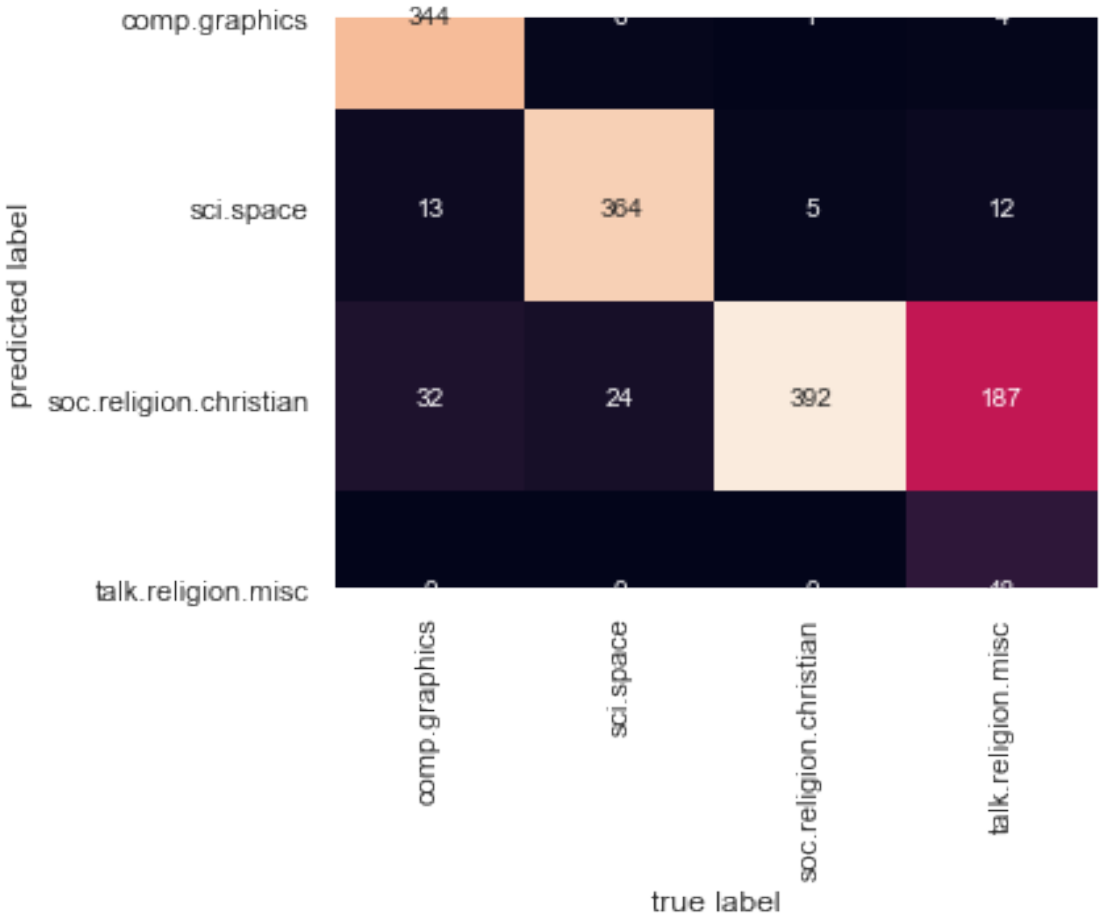
```
['comp.graphics', 'sci.space', 'soc.religion.christian', 'talk.religion.misc']
```

[72]:
```
#TF-IDF
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
```

```
from sklearn.pipeline import make_pipeline

model = make_pipeline(TfidfVectorizer(), MultinomialNB())

model.fit(train.data, train.target)
labels = model.predict(test.data)
```

[59]:
```
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(test.target, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=train.target_names, yticklabels=train.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label')
```

[59]:
```
Text(52.893125000000005, 0.5, 'predicted label')
```



[67]:
```
def predict_category(s, train=train, model=model):
    pred = model.predict([s])
    return train.target_names[pred[0]]
```

```
predict_category('sending a payload to the ISS')
```

[67]: 'sci.space'

[78]:
```
predict_category('disscussing islam vs atheism')
```

[78]: 'soc.religion.christian'

[80]:
```
predict_category('determining the screen resolution')
```

[80]: 'comp.graphics'

[ ]:
```

```

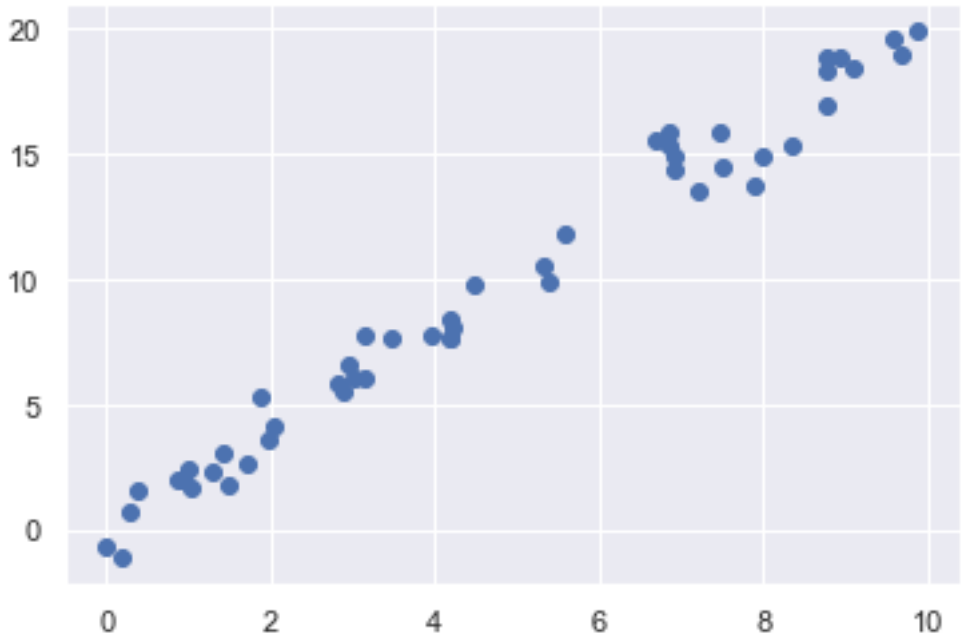
??_????

July 8, 2019

[36]:
```
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np

rng = np.random.RandomState(1)
x = 10 * rng.rand(50)
y = 2 * x + rng.randn(50)
plt.scatter(x, y)
```

```
(50,)
(50,)
```

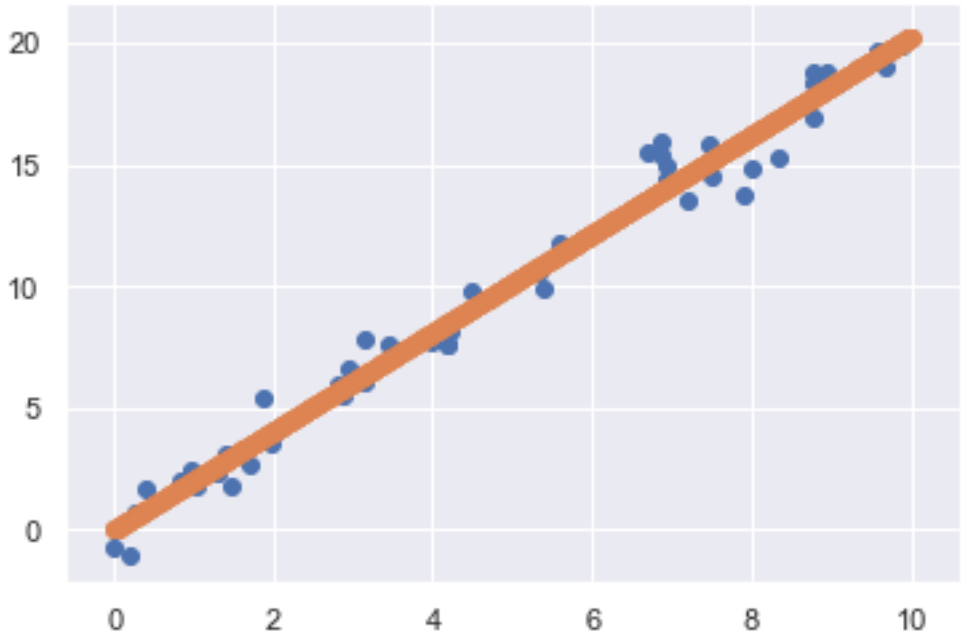
[36]:
```
<matplotlib.collections.PathCollection at 0x12b9adc88>
```



[37]:
```
#LinearRegression
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)
model.fit(x[:, np.newaxis], y)
xfit = np.linspace(0, 10, 1000)
yfit = model.predict(xfit[:, np.newaxis])

plt.scatter(x, y)
plt.scatter(xfit, yfit)
```

```
(50,)
```

[37]:
```
<matplotlib.collections.PathCollection at 0x12bc8def0>
```



[38]:
```
print("Model slope: ", model.coef_[0])
print("Model intercept: ", model.intercept_)
```

```
Model slope:  2.0272088103606953
Model intercept:  0.0014229144467954313
```

[34]:
```
#
#
rng = np.random.RandomState(1)
X = 10 * rng.rand(100, 3)
y = 0.5 + np.dot(X, [1.5, -2, 3])
```

```
model.fit(X, y)
print(model.intercept_)
print(model.coef_)
```

```
(100, 3)
(100,)
0.5000000000000036
[ 1.5 -2.   3. ]
```

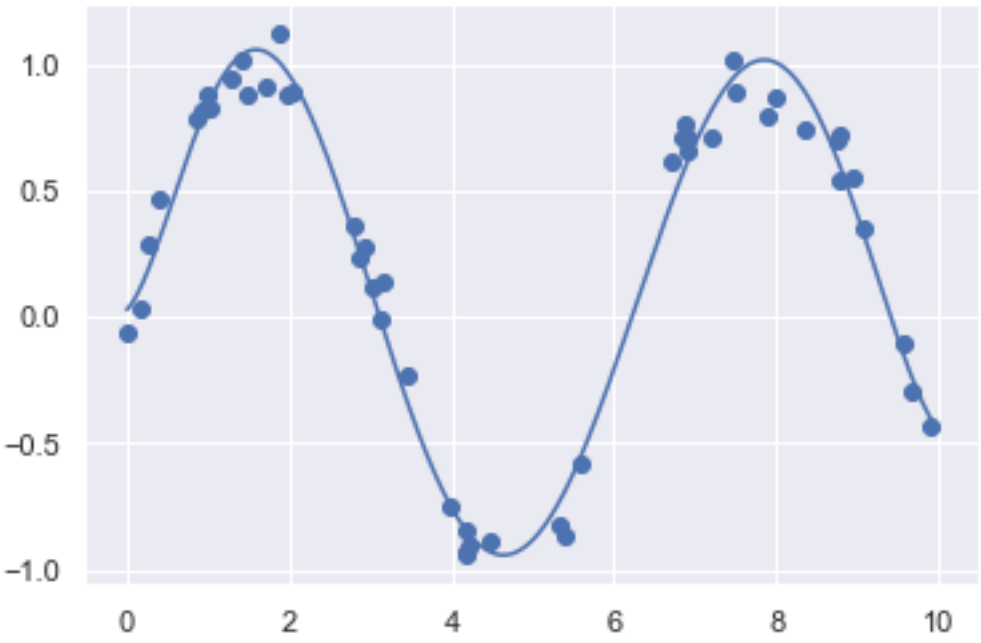
1

[15]:
```
from sklearn.preprocessing import PolynomialFeatures
x = np.array([2, 3, 4])
poly = PolynomialFeatures(3, include_bias=False)
poly.fit_transform(x[:, None])
```

[15]:
```
array([[ 2.,  4.,  8.],
       [ 3.,  9., 27.],
       [ 4., 16., 64.]])
```

[30]:
```
from sklearn.pipeline import make_pipeline
#7
poly_model = make_pipeline(PolynomialFeatures(7),
                           LinearRegression())
rng = np.random.RandomState(1)
x = 10 * rng.rand(50)
y = np.sin(x) + 0.1 * rng.randn(50)

poly_model.fit(x[:, None], y)
xfit = np.linspace(0, 10, 1000)
yfit = poly_model.predict(xfit[:, None])

plt.scatter(x, y)
plt.plot(xfit, yfit)
```

[30]:
```
[<matplotlib.lines.Line2D at 0x12b89b208>]
```



2

[ ]:

??_??????

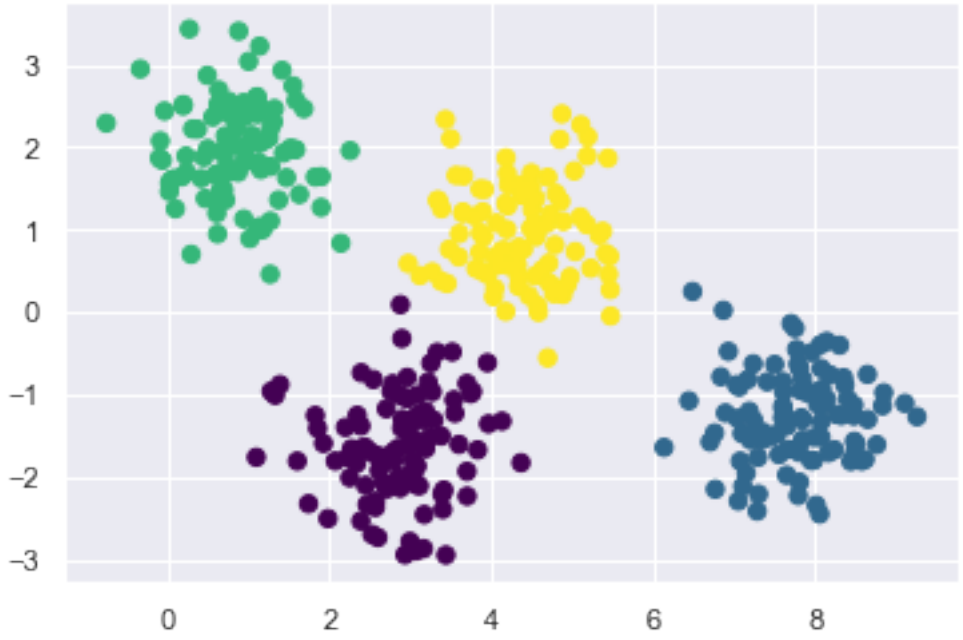
July 8, 2019

[5]:
```
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
```

[6]:
```
#
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=400, centers=4, cluster_std=0.60,␣
 ↪random_state=0)
X = X[:, ::-1]#
```

[7]:
```
#k-means
from sklearn.cluster import KMeans
kmeans = KMeans(4, random_state=0)
labels = kmeans.fit(X).predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis')
```

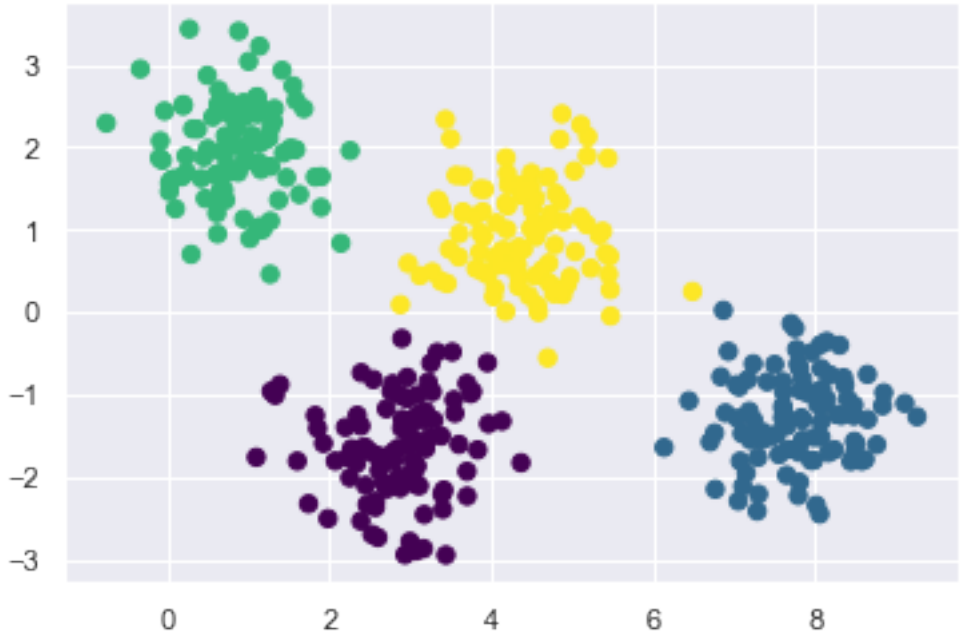
[7]:
```
<matplotlib.collections.PathCollection at 0x12b201dd8>
```



1

[10]:
```
#GMMk-means
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=4).fit(X)
labels = gmm.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis')
```

[10]:
```
<matplotlib.collections.PathCollection at 0x12d264d68>
```

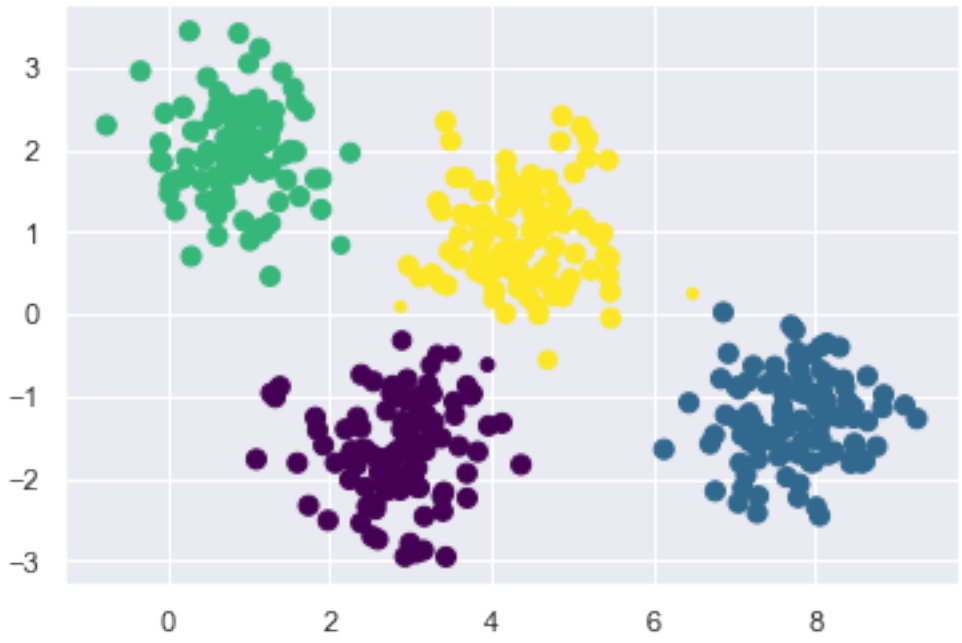


[11]:
```
probs = gmm.predict_proba(X)
print(probs[:5].round(3))
```

```
[[0.    0.469 0.    0.531]
 [1.    0.    0.    0.   ]
 [1.    0.    0.    0.   ]
 [0.    0.    0.    1.   ]
 [1.    0.    0.    0.   ]]
```

[12]:
```
size = 50 * probs.max(1) ** 2
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=size)
```

[12]:
```
<matplotlib.collections.PathCollection at 0x12f413198>
```

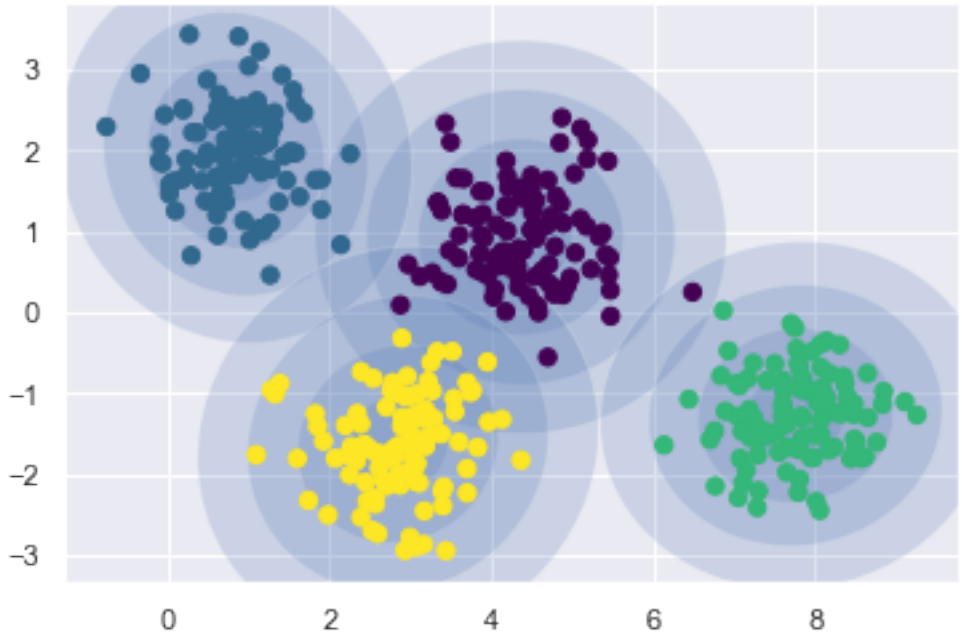


[48]:
```
#GMMgmm
from matplotlib.patches import Ellipse
def draw_ellipse(position, covariance, ax=None, **kwargs):
    """"""
    ax = ax or plt.gca()
    #
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

    #
    for nsig in range(1, 4):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height, angle,␣
 ↪**kwargs))

def plot_gmm(gmm, X, label=True, ax=None):
    ax = ax or plt.gca()
    labels = gmm.fit(X).predict(X)
    if label:
        ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis', zorder=2)
    else:
        ax.scatter(X[:, 0], X[:, 1], s=40, zorder=2)
```

```
    ax.axis('equal')

    w_factor = 0.2 / gmm.weights_.max()
    print(gmm.means_)
    print(gmm.covariances_)
    print(gmm.weights_)
    for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
        draw_ellipse(pos, covar, alpha=w * w_factor)
```

[49]:
```
gmm = GaussianMixture(n_components=4, random_state=42)
plot_gmm(gmm, X)
```

```
[[ 4.36221851  0.93141671]
 [ 0.83760086  1.9579654 ]
 [ 7.76222624 -1.27635633]
 [ 2.84407337 -1.62828783]]
[[[ 0.40398685 -0.00235346]
  [-0.00235346  0.36933794]]

 [[ 0.29768676 -0.02444193]
  [-0.02444193  0.34173529]]

 [[ 0.36530305  0.01293533]
  [ 0.01293533  0.28921965]]

 [[ 0.38465995  0.02716798]
  [ 0.02716798  0.37394003]]]
[0.25454734 0.24963804 0.24878736 0.24702726]
```



[ ]:

[ ]:

?????

July 8, 2019

[29]:
```
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))
```

[30]:
```
import numpy as np
def make_data(N, err=1.0, rseed=1):
    #
    rng = np.random.RandomState(rseed)
    X = rng.rand(N, 1) ** 2
    print(X)
    y = 10 - 1. / (X.ravel() + 0.1)
    print()
    print(y)
    if err > 0:
        y += err * rng.randn(N)
    return X, y
X, y = make_data(42)
```

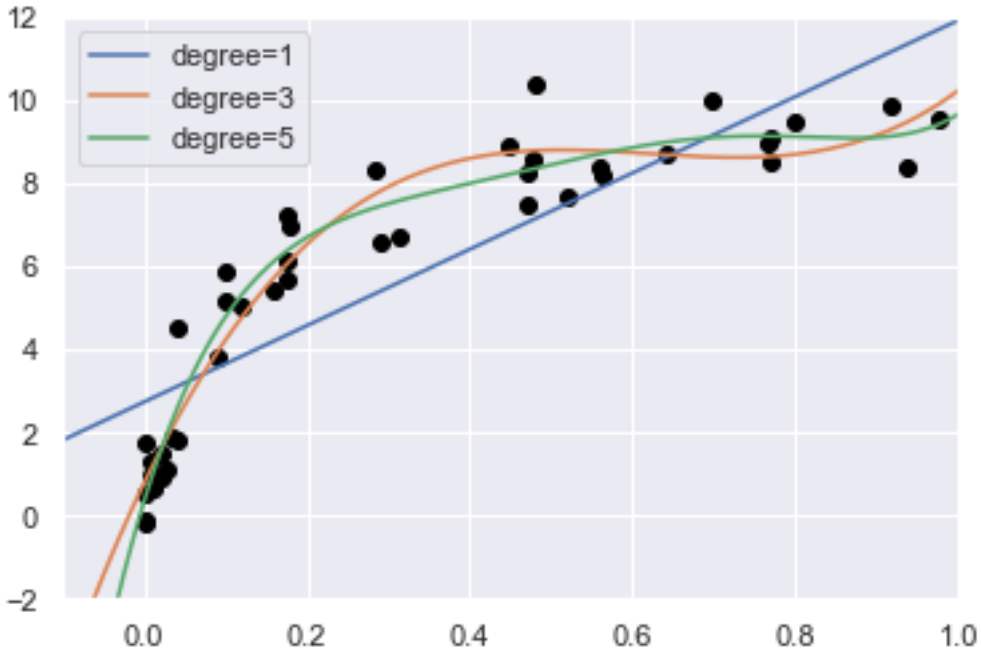
```
[[1.73907352e-01]
 [5.18867376e-01]
 [1.30815988e-08]
 [9.14049845e-02]
 [2.15372915e-02]
 [8.52641608e-03]
 [3.46928663e-02]
 [1.19412216e-01]
 [1.57424429e-01]
 [2.90323473e-01]
 [1.75724041e-01]
 [4.69525764e-01]
 [4.18007224e-02]
 [7.71090232e-01]
 [7.50080261e-04]
```

```
 [4.49526682e-01]
 [1.74143298e-01]
 [3.12134324e-01]
 [1.97084925e-02]
 [3.92442000e-02]
 [6.41191864e-01]
 [9.37530479e-01]
 [9.82347155e-02]
 [4.79310604e-01]
 [7.68057946e-01]
 [8.00321082e-01]
 [7.23251789e-03]
 [1.52527609e-03]
 [2.88423714e-02]
 [7.71134256e-01]
 [9.67209972e-03]
 [1.77331632e-01]
 [9.17552352e-01]
 [2.84265221e-01]
 [4.78693941e-01]
 [9.95501134e-02]
 [4.71283524e-01]
 [6.96600012e-01]
 [3.34461088e-04]
 [5.62716493e-01]
 [9.77846253e-01]
 [5.59751846e-01]]

[6.34913050e+00 8.38414491e+00 1.30815971e-06 4.77547566e+00
 1.77207269e+00 7.85653520e-01 2.57570184e+00 5.44236863e+00
 6.11536479e+00 7.43802239e+00 6.37318532e+00 8.24415318e+00
 2.94784975e+00 8.85201330e+00 7.44495944e-02 8.18025215e+00
 6.35227267e+00 7.57360661e+00 1.64637380e+00 2.81837233e+00
 8.65082167e+00 9.03617289e+00 4.95547489e+00 8.27381030e+00
 8.84800317e+00 8.88928515e+00 6.74470583e-01 1.50236094e-01
 2.23857812e+00 8.85207132e+00 8.81910691e-01 6.39420865e+00
 9.01724958e+00 7.39763074e+00 8.27197085e+00 4.98872748e+00
 8.24955568e+00 8.74466484e+00 3.33346175e-02 8.49105913e+00
 9.07222389e+00 8.48427859e+00]
```

[31]:
```
import matplotlib.pyplot as plt
import seaborn; seaborn.set()#

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

plt.scatter(X.ravel(), y, color='black')
axis = plt.axis()
```

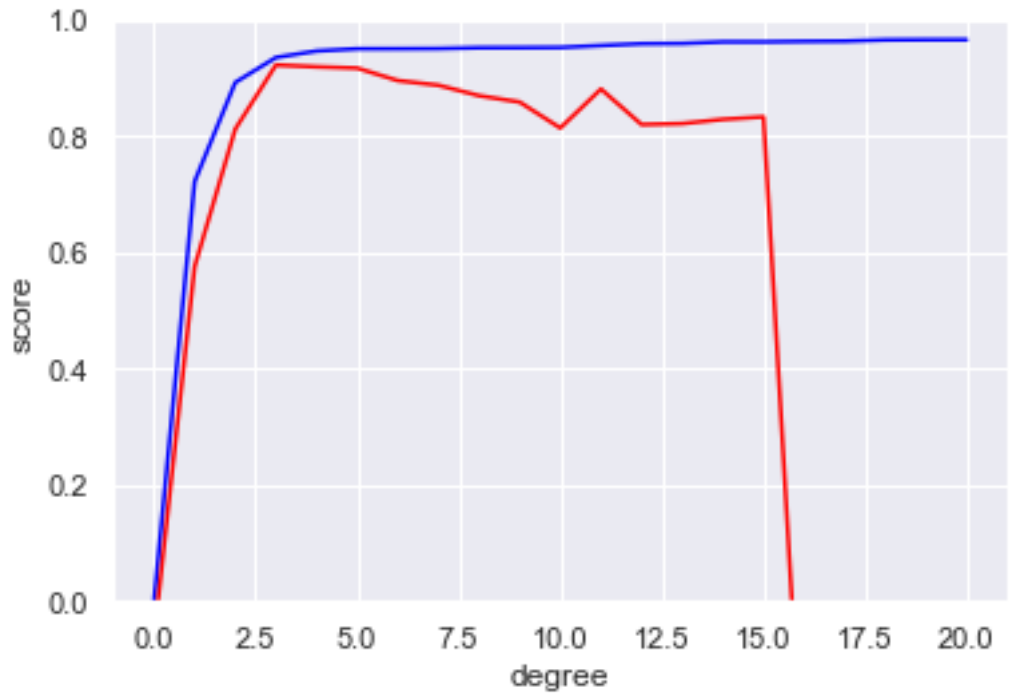
```
for degree in [1, 3, 5]:
    y_test = PolynomialRegression(degree).fit(X, y).predict(X_test)
    plt.plot(X_test.ravel(), y_test, label='degree={0}'.format(degree))
plt.xlim(-0.1, 1.0)
plt.ylim(-2, 12)
plt.legend(loc='best')
```

[31]: <matplotlib.legend.Legend at 0x11eb4b048>



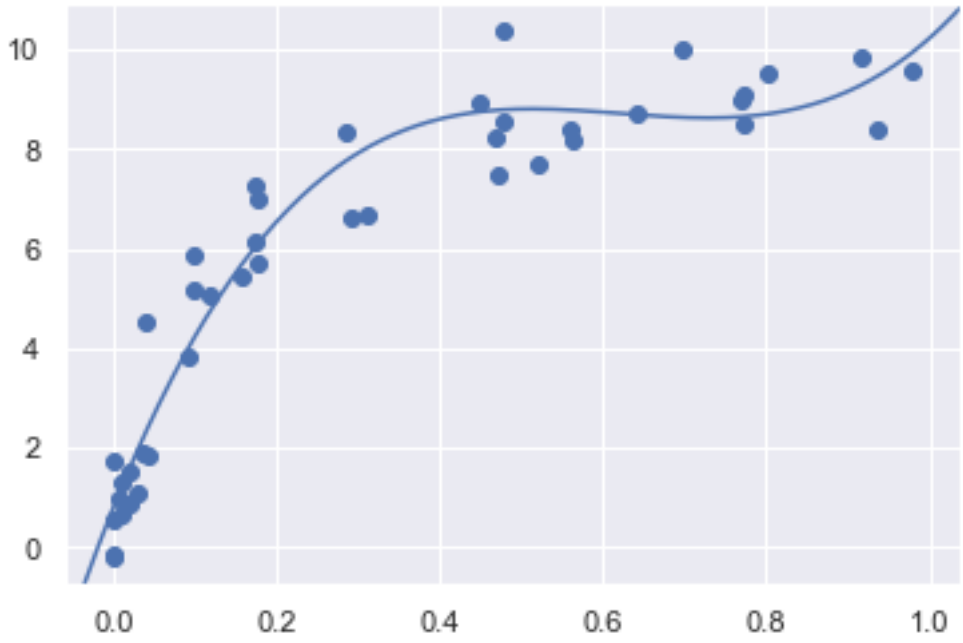
[33]:
```
#()()
#
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import validation_curve
degree = np.arange(0, 21)
train_score, val_score = validation_curve(PolynomialRegression(), X, y,
                                          'polynomialfeatures__degree',
                                          degree, cv=7)
plt.plot(degree, np.median(train_score, 1), color='blue', label='training␣
 ↪score')
plt.plot(degree, np.median(val_score, 1), color='red', label='validation score')
plt.ylim(0, 1)
plt.xlabel('degree')
plt.ylabel('score')
```

[33]: Text(0, 0.5, 'score')



[39]:
```
#
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)
```

[39]:
```
(-0.0587489297463609,
 1.036595195981298,
 -0.7459943120970807,
 10.918045992764213)
```


10
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0

[ ]:

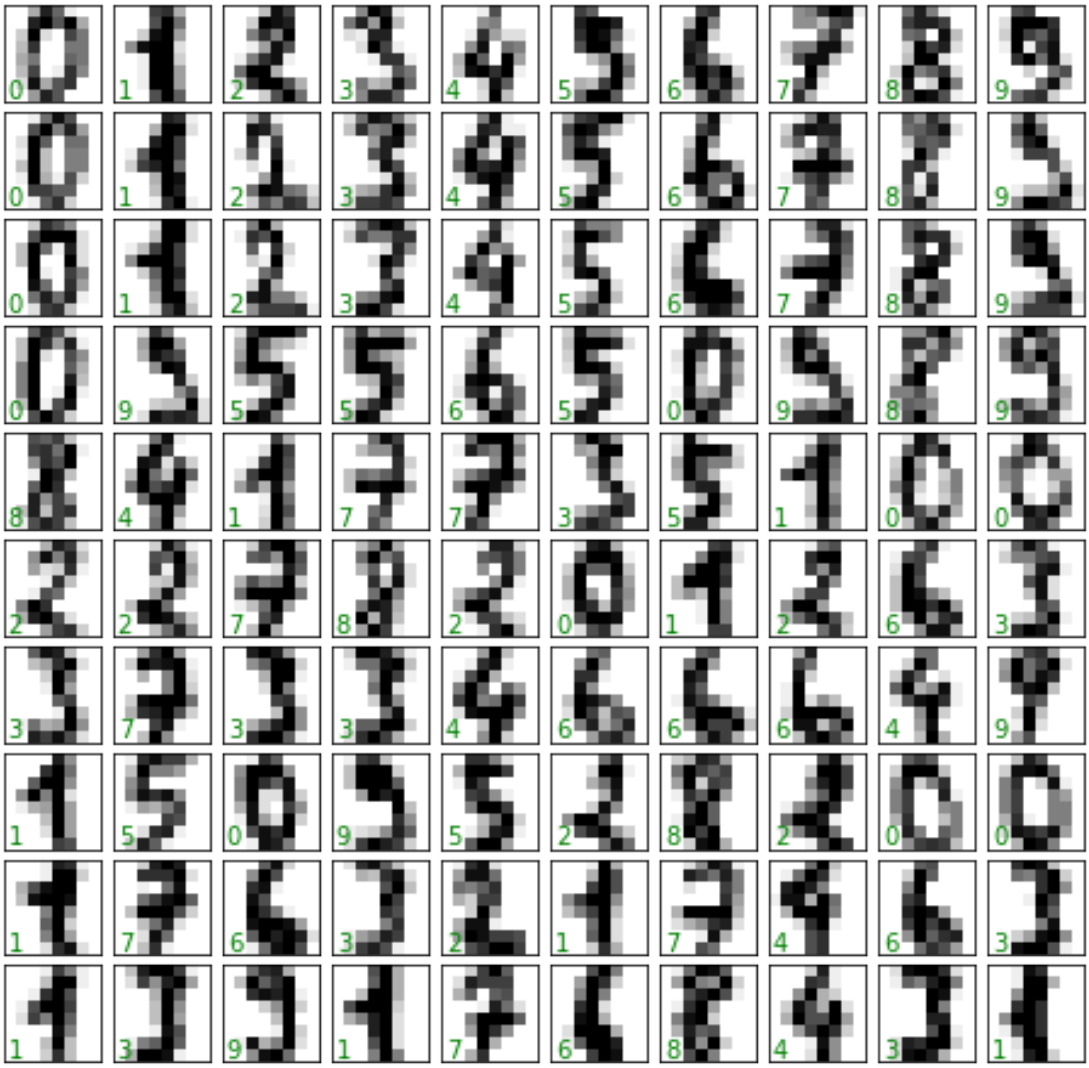
??????

July 8, 2019

[1]:
```
from sklearn.datasets import load_digits
digits = load_digits()
digits.images.shape
```

[1]:
```
(1797, 8, 8)
```

[30]:
```
#100
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')
```



[31]:
```
X = digits.data
X.shape
```

[31]:
```
(1797, 64)
```

[32]:
```
y = digits.target
y.shape
```

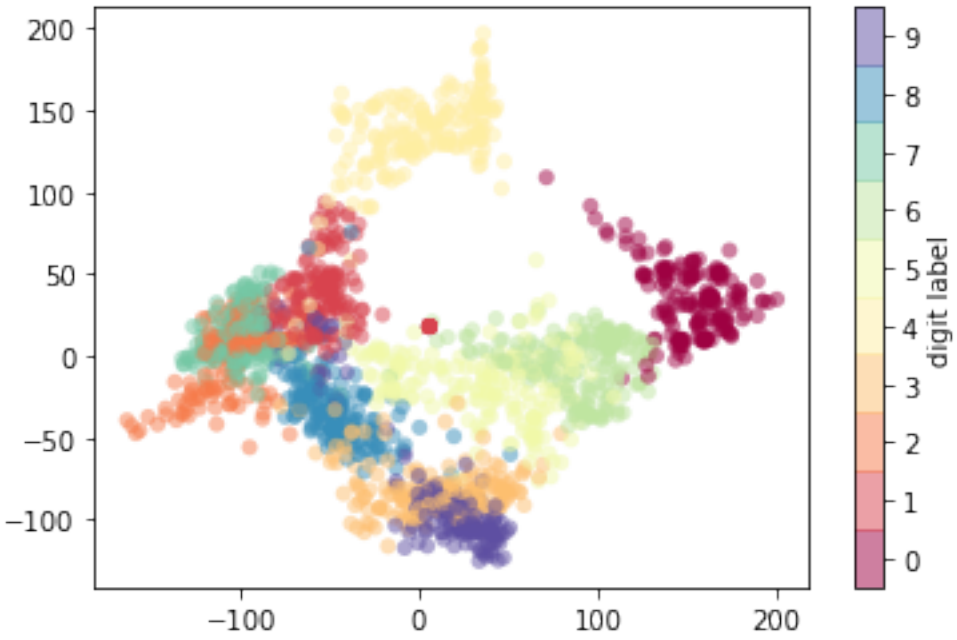
[32]:
```
(1797,)
```

1

[35]:
```
#Isomap
from sklearn.manifold import Isomap
iso = Isomap(n_components=2)
iso.fit(digits.data)
data_projected = iso.transform(digits.data)
data_projected.shape
```

[35]:
```
(1797, 2)
```

[44]:
```
import matplotlib.pyplot as plt
plt.scatter(data_projected[:, 0], data_projected[:, 1], c=digits.target,
            edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('Spectral', 10))
plt.colorbar(label='digit label', ticks=range(10))
plt.clim(-0.5, 9.5)
```



2

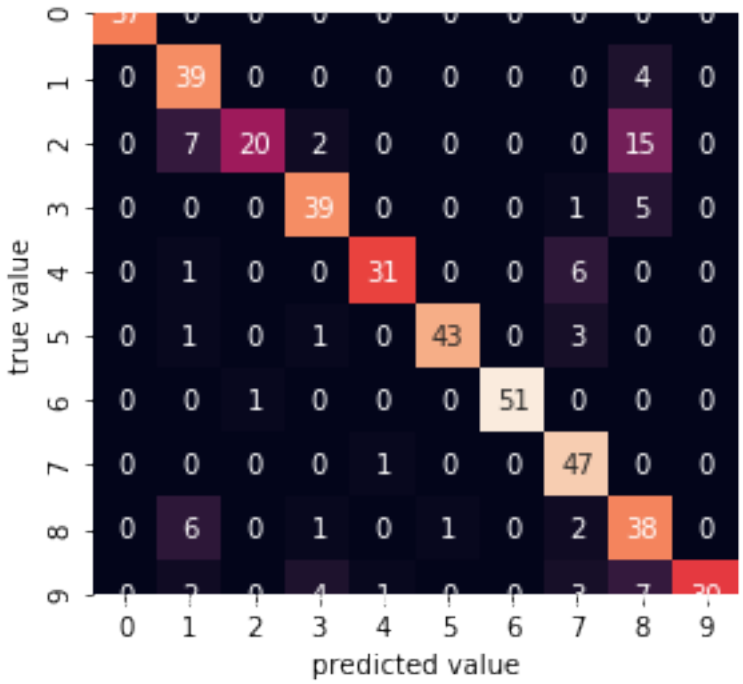
[46]:
```
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=0)
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(Xtrain, ytrain)
y_model = model.predict(Xtest)
```

[47]:
```
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
```

[47]:
```
0.8333333333333334
```

[53]:
```
#
import seaborn as sns
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(ytest, y_model)
sns.heatmap(mat, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value')
```

[53]:
```
Text(79.60000000000001, 0.5, 'true value')
```



[ ]:

????

July 8, 2019

[3]:
```
#:
data = [{'price': 850000, 'rooms': 4, 'neighborhood': 'Queen Anne'},
       {'price': 700000, 'rooms': 3, 'neighborhood': 'Fremont'},
       {'price': 650000, 'rooms': 3, 'neighborhood': 'Wallingford'},
       {'price': 600000, 'rooms': 2, 'neighborhood': 'Fremont'}]

#
#01
from sklearn.feature_extraction import DictVectorizer
vec = DictVectorizer(sparse=False, dtype=int)
vec.fit_transform(data)
```

[3]:
```
array([[     0,      1,      0, 850000,      4],
       [     1,      0,      0, 700000,      3],
       [     0,      0,      1, 650000,      3],
       [     1,      0,      0, 600000,      2]])
```

[4]:
```
#
vec.get_feature_names()
```

[4]:
```
['neighborhood=Fremont',
 'neighborhood=Queen Anne',
 'neighborhood=Wallingford',
 'price',
 'rooms']
```

[5]:
```
from sklearn.feature_extraction import DictVectorizer
#
vec = DictVectorizer(sparse=True, dtype=int)
vec.fit_transform(data)
```

[5]:
```
<4x5 sparse matrix of type '<class 'numpy.int64'>'
        with 12 stored elements in Compressed Sparse Row format>
```

[8]:
```
sample = ["problem of evil",
          "evil queen",
          "horizon problem"]
#Scikit-LearnCountVector
#
from sklearn.feature_extraction.text import CountVectorizer
```

```
vec = CountVectorizer()
X = vec.fit_transform(sample)
#DataFrame
import pandas as pd
pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
```

[8]:

	evil	horizon	of	problem	queen
0	1	0	1	1	0
1	1	0	0	0	1
2	0	1	0	1	0

[10]:
```
#
from sklearn.feature_extraction.text import TfidfVectorizer
vec = TfidfVectorizer()
X = vec.fit_transform(sample)
pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
```

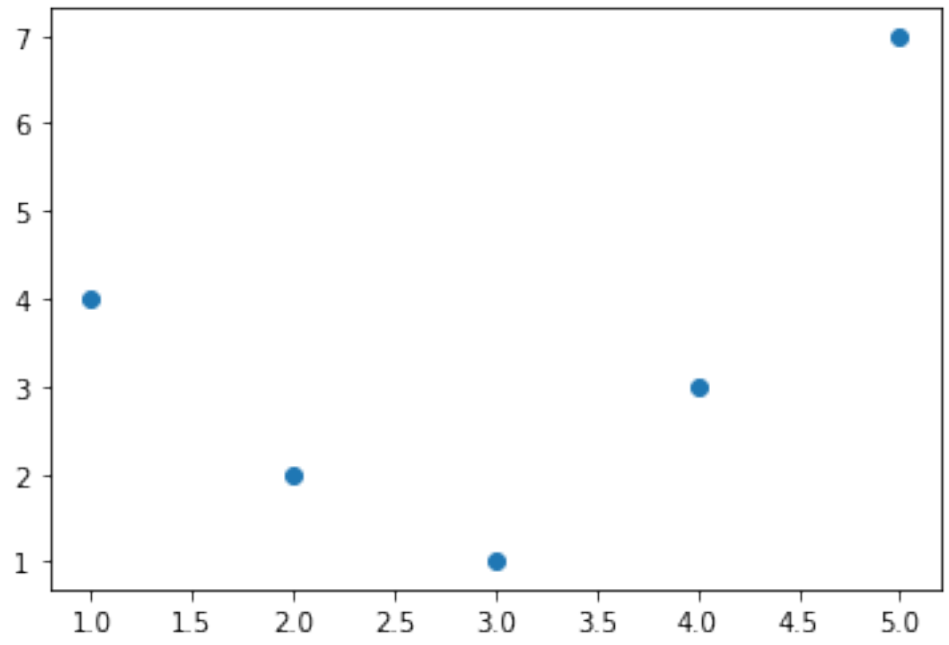
[10]:

	evil	horizon	of	problem	queen
0	0.517856	0.000000	0.680919	0.517856	0.000000
1	0.605349	0.000000	0.000000	0.000000	0.795961
2	0.000000	0.795961	0.000000	0.605349	0.000000

1

[12]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1, 2, 3, 4, 5])
y = np.array([4, 2, 1, 3, 7])
plt.scatter(x, y)
```

[12]:
```
<matplotlib.collections.PathCollection at 0x12dbfe278>
```

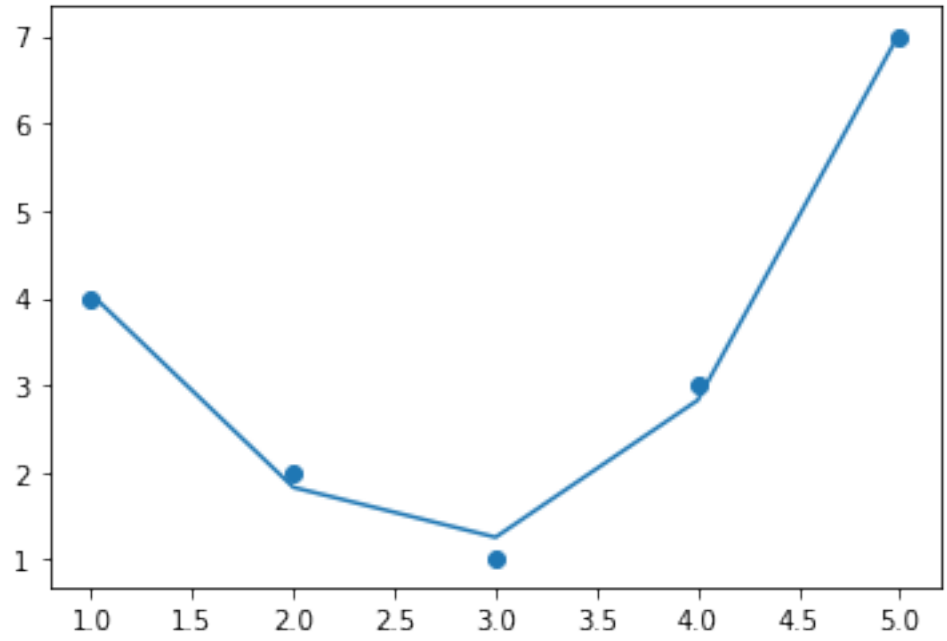


[19]:
```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=3, include_bias=False)
X = x[:, np.newaxis]
X2 = poly.fit_transform(X)
print(X2)
```

```
[[  1.   1.   1.]
 [  2.   4.   8.]
 [  3.   9.  27.]
 [  4.  16.  64.]
 [  5.  25. 125.]]
```

[20]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(X2, y)
yfit = model.predict(X2)
plt.scatter(x, y)
plt.plot(x, yfit)
```

[20]:
```
[<matplotlib.lines.Line2D at 0x12dd2bf60>]
```


7
6
5
4
3
2
1
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

[ ]:

?????

July 8, 2019

[3]:
```
import numpy as np
from numpy import nan
X = np.array([[nan, 0, 3],
              [3, 7, 9],
              [3, 5, 2],
              [4, nan, 6],
              [8, 8, 1]])
y = np.array([14, 16, -1, 8, -5])
```

1

[4]:
```
#
from sklearn.preprocessing import Imputer
imp = Imputer(strategy='mean')
X2 = imp.fit_transform(X)
X2
```

```
/usr/local/miniconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:66: DeprecationWarning: Class Imputer is
deprecated; Imputer was deprecated in version 0.20 and will be removed in 0.22.
Import impute.SimpleImputer from sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

[4]:
```
array([[4.5, 0. , 3. ],
       [3. , 7. , 9. ],
       [3. , 5. , 2. ],
       [4. , 5. , 6. ],
       [8. , 8. , 1. ]])
```

2

[8]:
```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
```

```
model = make_pipeline(Imputer(strategy='mean'),
                      PolynomialFeatures(degree=2),
                      LinearRegression())

model.fit(X, y)
print(y)
print(model.predict(X))
```

```
[14 16 -1  8 -5]
[14. 16. -1.  8. -5.]
```

```
/usr/local/miniconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:66: DeprecationWarning: Class Imputer is
deprecated; Imputer was deprecated in version 0.20 and will be removed in 0.22.
Import impute.SimpleImputer from sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

[ ]:

???

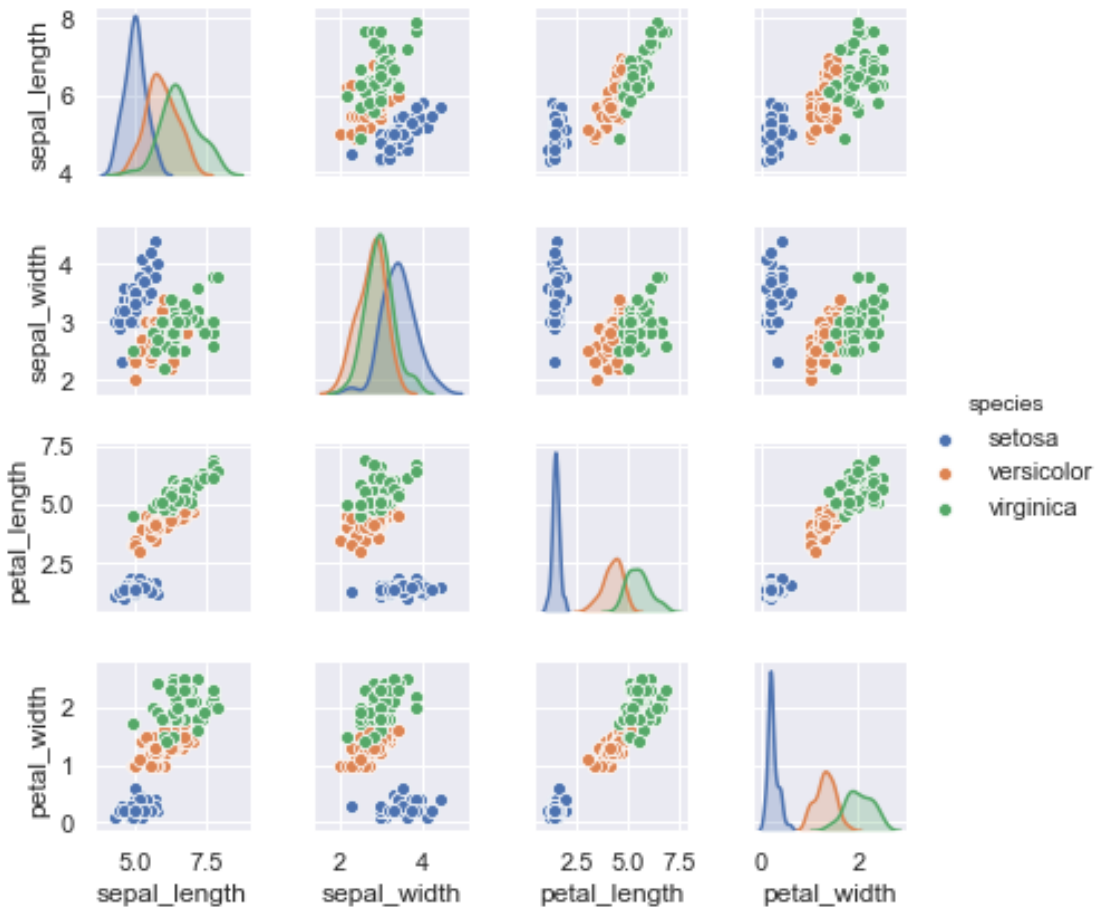
July 8, 2019

[51]:
```
import seaborn as sns
iris = sns.load_dataset('iris')
iris.head()
```

[51]:

	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

[52]:
```
import matplotlib as plt
import seaborn as sns; sns.set()
iris = sns.load_dataset("iris")
plt.rcParams["text.usetex"] = False
g = sns.pairplot(iris, hue='species', height=1.5)
```



[53]:
```
#
X_iris = iris.drop('species', axis=1)
X_iris.shape
```

[53]:
```
(150, 4)
```

[54]:
```
y_iris = iris['species']
y_iris.shape
```

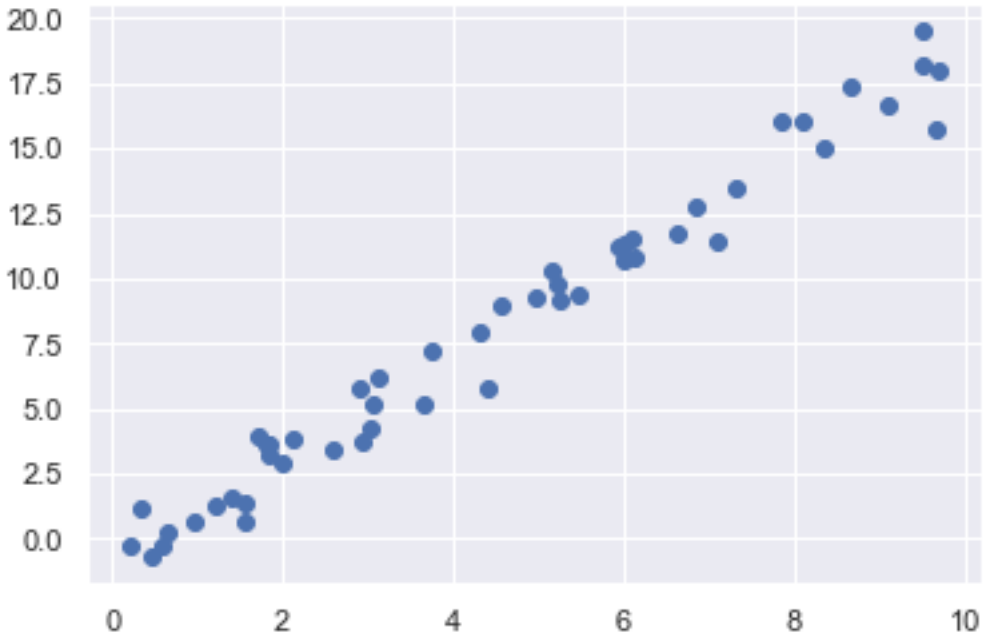
[54]:
```
(150,)
```

[55]:
```
#Scikit-LearnAPI:
#(1)Scikit_Learn
#(2)
#(3)
#(4)fit()
#(5) (predict())
#               (transform()predict())
```

1

[56]:
```
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)
```

[56]:
```
<matplotlib.collections.PathCollection at 0x12e7bb7b8>
```



[57]:
```
#Scikit-LearnPython
#
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model
```

[57]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

[58]:
```
X = x[:, np.newaxis]
Y = y[:, np.newaxis]
```

[59]:
```
model.fit(X, y)
```

[59]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

[60]:
```
model.coef_
```

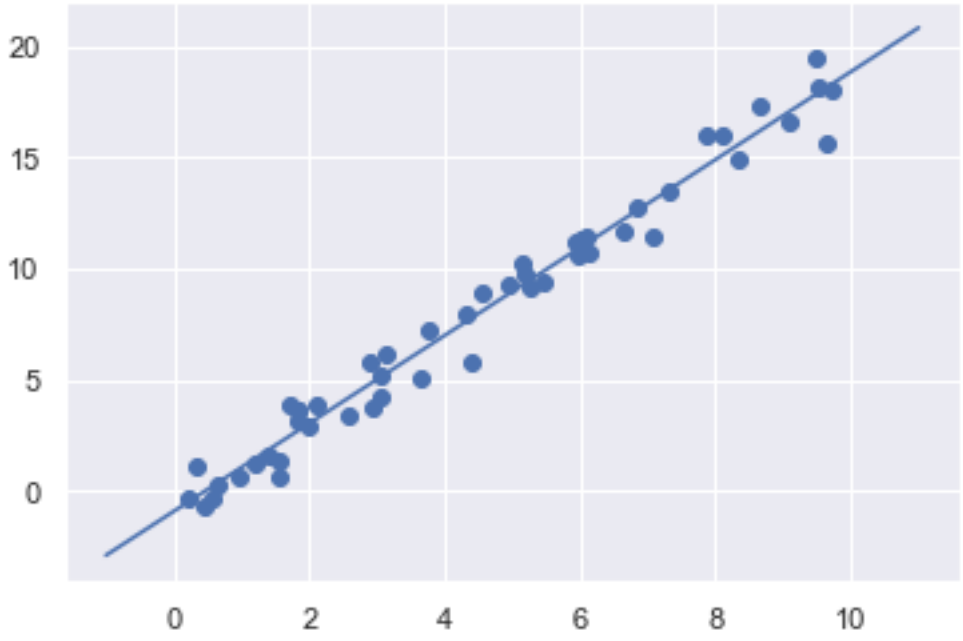
[60]:
```
array([1.9776566])
```

[61]:
```
model.intercept_
```

[61]:
```
-0.9033107255311146
```

[62]:
```
xfit = np.linspace(-1, 11)
Xfit = xfit[:, np.newaxis]
yfit = model.predict(Xfit)
plt.scatter(x, y)
plt.plot(xfit, yfit)
```

[62]:
```
[<matplotlib.lines.Line2D at 0x12a46bba8>]
```



2

[75]:
```
#train_test_split
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X_iris, y_iris, random_state=1)
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(Xtrain, ytrain)
y_model = model.predict(Xtest)
```

[76]:
```
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
```

[76]:
```
0.9736842105263158
```

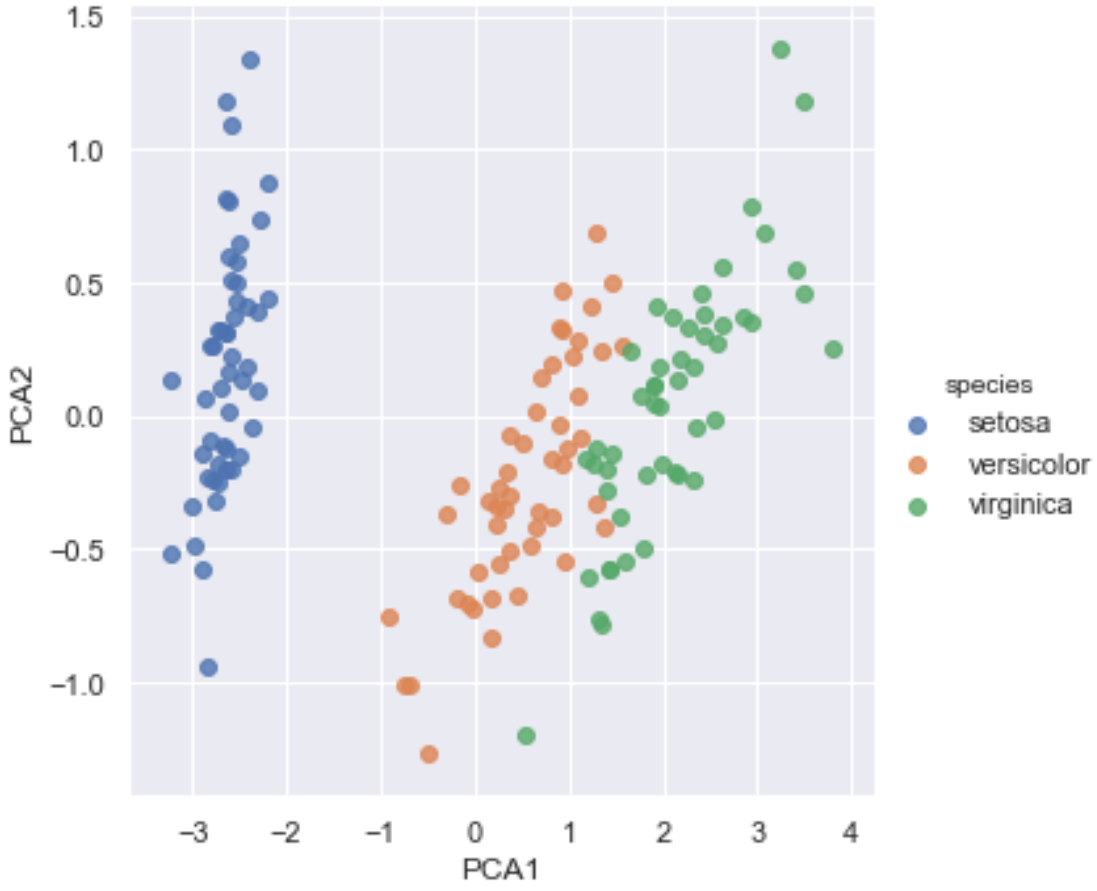
[77]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(model, X_iris, y_iris, cv=5)
```

[77]:
```
array([0.93333333, 0.96666667, 0.93333333, 0.93333333, 1.        ])
```

3

[65]:
```
from sklearn.decomposition import PCA
model = PCA(n_components=2)
model.fit(X_iris)
X_2D = model.transform(X_iris)

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

[65]:
```
<seaborn.axisgrid.FacetGrid at 0x12e6f9978>
```

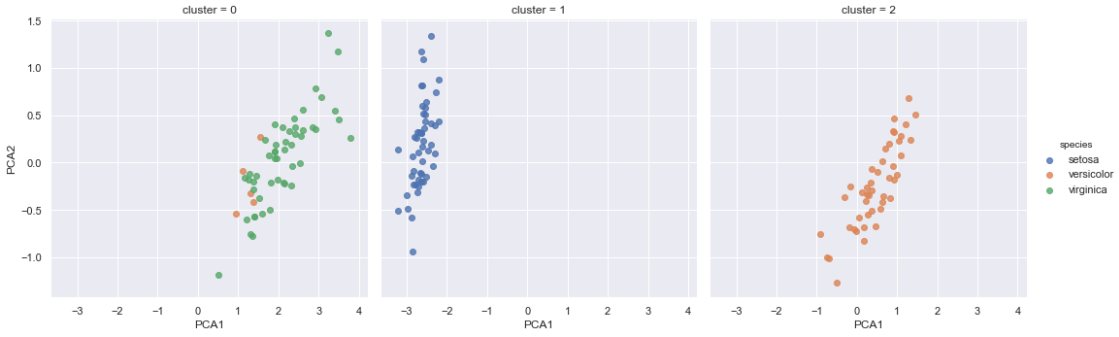


4

[70]:
```
from sklearn.mixture import GaussianMixture
model = GaussianMixture(n_components=3, covariance_type='full')
model.fit(X_iris)
y_gmm = model.predict(X_iris)
```

[71]:
```
iris['cluster'] = y_gmm
sns.lmplot("PCA1", "PCA2", hue = 'species', data = iris, col='cluster',␣
 ↪fit_reg=False)
```

[71]:
```
<seaborn.axisgrid.FacetGrid at 0x12c69c198>
```



[ ]:
```

```