In [1]:

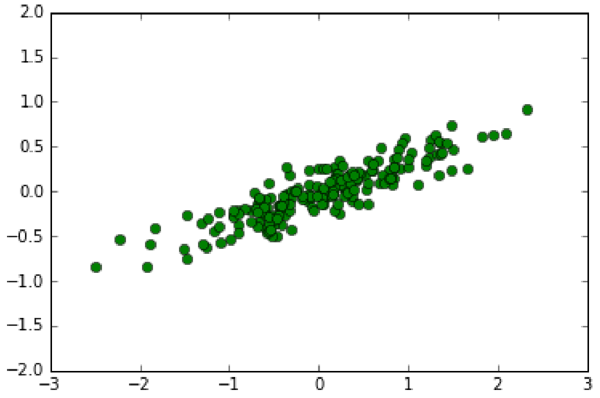
```
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:

```
np.random.seed(1)
X = np.dot(np.random.random(size=(2, 2)), np.random.normal(size=(2, 200))).T
plt.plot(X[:, 0], X[:, 1], 'og')
plt.axis('equal')
```

Out[2]:

```
(-3.0, 3.0, -1.0, 1.0)
```



In [3]:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X)
```

Out[3]:

```
PCA(copy=True, n_components=2, whiten=False)
```

In [4]:

```
print(pca.explained_variance_ratio_)
```

```
[ 0.97634101  0.02365899]
```

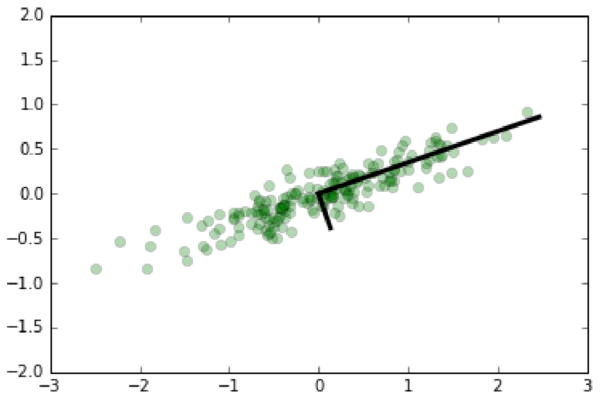
In [5]:

```
print(pca.components_)
```

```
[[ 0.94446029  0.32862557]
 [ 0.32862557 -0.94446029]]
```

In [6]:

```
plt.plot(X[:, 0], X[:, 1], 'og', alpha=0.3)
plt.axis('equal')
for length, vector in zip(pca.explained_variance_, pca.components_):
    v = vector * 3 * np.sqrt(length)
    plt.plot([0, v[0]], [0, v[1]], '-k', lw=3)
```



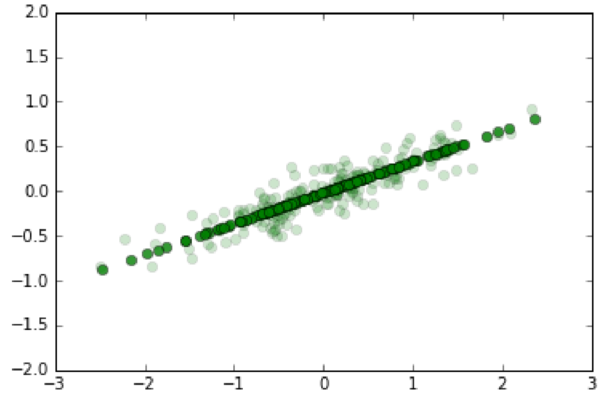
In [7]:

```
clf = PCA(0.97)
X_trans = clf.fit_transform(X)
print(X.shape)
print(X_trans.shape)
```

```
(200, 2)
(200, 1)
```

In [8]:

```
X_new = clf.inverse_transform(X_trans)
plt.plot(X[:, 0], X[:, 1], 'og', alpha=0.2)
plt.plot(X_new[:, 0], X_new[:, 1], 'og', alpha=0.8)
plt.axis('equal');
```

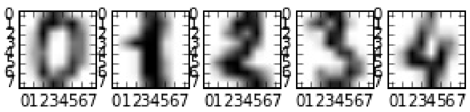


In [9]:

```
from sklearn.datasets import load_digits
digits = load_digits()
X = digits.data
y = digits.target
```

In [10]:

```
plt.rc("image", cmap="binary")  # this sets a black on white colormap
for i in range(0,5):
    plt.subplot(1, 6, 2 + i)
    plt.imshow(X[i,:].reshape(8, 8))
```



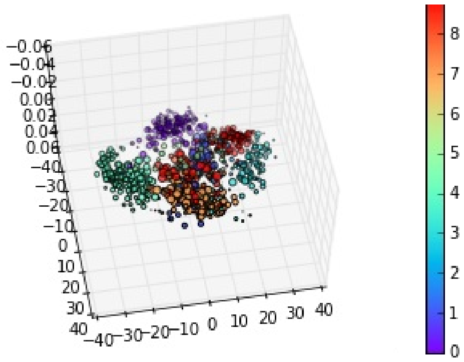
In [11]:

```
pca = PCA(3)  # project from 64 to 3 dimensions
Xproj = pca.fit_transform(X)
print(X.shape)
print(Xproj.shape)
```

```
(1797, 64)
(1797, 3)
```

In [14]:

```
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
plt.rc("image", cmap="rainbow")
ax = fig.add_subplot(111, projection='3d')
ax.view_init(300,100)
plt.scatter(Xproj[:, 0], Xproj[:, 1], Xproj[:, 2], c=y)
plt.colorbar();
```



In [15]:

```
from sklearn.linear_model import LogisticRegression

cutPoint = int(len(X)/2) #Find out the midpoint in the data set

#as usual split the data into a training set and they set, do that for the original data set and for t
he projected data set
X_train = X[:cutPoint]
y_train = y[:cutPoint]
X_test = X[cutPoint:]
y_test = y[cutPoint:]
Xproj_train = Xproj[:cutPoint]
Xproj_test = Xproj[cutPoint:]

#Create logistic regression model with the original data
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

#Create a logistic regression model with the projected data (obtained from PCA)
logregproj = LogisticRegression()
logregproj.fit(Xproj_train, y_train)

print "Accuracy on test set using original data:", logreg.score(X_test, y_test)
print "Accuracy on test set using projected data:", logregproj.score(Xproj_test, y_test)
```

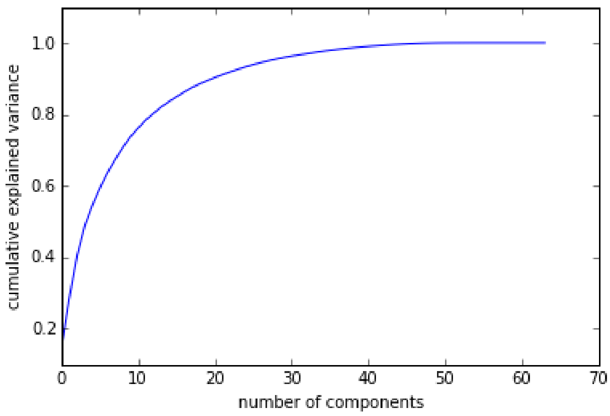
```
Accuracy on test set using original data: 0.916573971079
Accuracy on test set using projected data: 0.654060066741
```

In [16]:

```
pca = PCA(64).fit(X)
#plt.semilogx(np.cumsum(pca.explained_variance_ratio_))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
```

Out[16]:

```
<matplotlib.text.Text at 0x8d1a590>
```



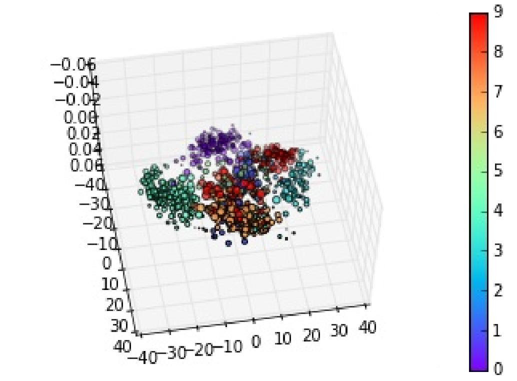
In [17]:

```
pca = PCA(10)  # project from 64 to 3 dimensions
Xproj = pca.fit_transform(X)
print(X.shape)
print(Xproj.shape)
```

```
(1797, 64)
(1797, 10)
```

In [18]:

```
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
plt.rc("image", cmap="rainbow")
ax = fig.add_subplot(111, projection='3d')
ax.view_init(300,100)
plt.scatter(Xproj[:, 0], Xproj[:, 1], Xproj[:, 2], c=y)
plt.colorbar();
```



In [19]:

```
from sklearn.linear_model import LogisticRegression

cutPoint = int(len(X)/2) #Find out the midpoint in the data set

#as usual split the data into a training set and they set, do that for the original data set and for t
he projected data set
X_train = X[:cutPoint]
y_train = y[:cutPoint]
X_test = X[cutPoint:]
y_test = y[cutPoint:]
Xproj_train = Xproj[:cutPoint]
Xproj_test = Xproj[cutPoint:]

#Create logistic regression model with the original data
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

#Create a logistic regression model with the projected data (obtained from PCA)
logregproj = LogisticRegression()
logregproj.fit(Xproj_train, y_train)

print "Accuracy on test set using original data:", logreg.score(X_test, y_test)
print "Accuracy on test set using projected data:", logregproj.score(Xproj_test, y_test)
```

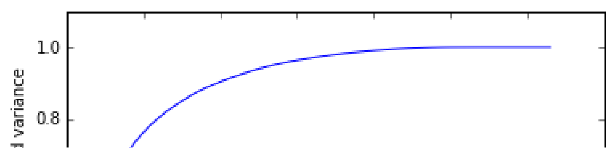
```
Accuracy on test set using original data: 0.916573971079
Accuracy on test set using projected data: 0.883203559511
```

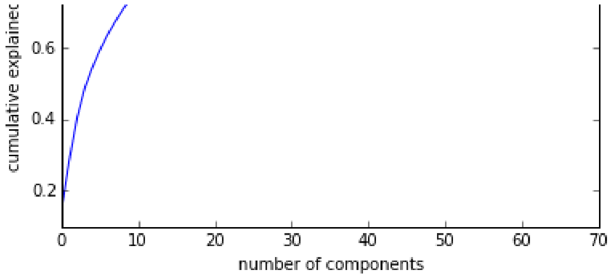
In [22]:

```
pca = PCA(64).fit(X)
#plt.semilogx(np.cumsum(pca.explained_variance_ratio_))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
```

Out[22]:

```
<matplotlib.text.Text at 0x8b661d0>
```





In [24]:

```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
cutPoint = int(len(X)/2) #Find out the midpoint in the data set

#as usual split the data into a training set and they set, do that for the original data set and for t
he projected data set
X_train = X[:cutPoint]
y_train = y[:cutPoint]
X_test = X[cutPoint:]
y_test = y[cutPoint:]
Xproj_train = Xproj[:cutPoint]
Xproj_test = Xproj[cutPoint:]

knn.fit(X_train, y_train)

#Create a logistic regression model with the projected data (obtained from PCA)
knnproj = KNeighborsClassifier(n_neighbors=3)
knn.fit(Xproj_train, y_train)

print "Accuracy on test set using original data:", knn.score(X_test, y_test)
print "Accuracy on test set using projected data:", knn.score(Xproj_test, y_test)
```

```
Accuracy on test set using original data:
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-24-853c7455e533> in <module>()
     17 knn.fit(Xproj_train, y_train)
     18
---> 19 print "Accuracy on test set using original data:", knn.score(X_test, y_test)
     20 print "Accuracy on test set using projected data:", knn.score(Xproj_test, y_test)

C:\Python27\lib\site-packages\sklearn\base.pyc in score(self, X, y, sample_weight)
    293         """
    294         from .metrics import accuracy_score
--> 295         return accuracy_score(y, self.predict(X), sample_weight=sample_weight)
    296
    297

C:\Python27\lib\site-packages\sklearn\neighbors\classification.pyc in predict(self, X)
    136         X = check_array(X, accept_sparse='csr')
    137
--> 138         neigh_dist, neigh_ind = self.kneighbors(X)
    139
    140         classes_ = self.classes_

C:\Python27\lib\site-packages\sklearn\neighbors\base.pyc in kneighbors(self, X, n_neighbors, return_dis
tance)
    372                     "or set algorithm='brute'" % self._fit_method)
    373             result = self._tree.query(X, n_neighbors,
--> 374                                       return_distance=return_distance)
    375         else:
    376             raise ValueError("internal: _fit_method not recognized")

C:\Python27\lib\site-packages\sklearn\neighbors\kd_tree.pyd in
sklearn.neighbors.kd_tree.BinaryTree.query (sklearn\neighbors\kd_tree.c:10454)()

ValueError: query data dimension must match training data dimension
```

In [ ]: