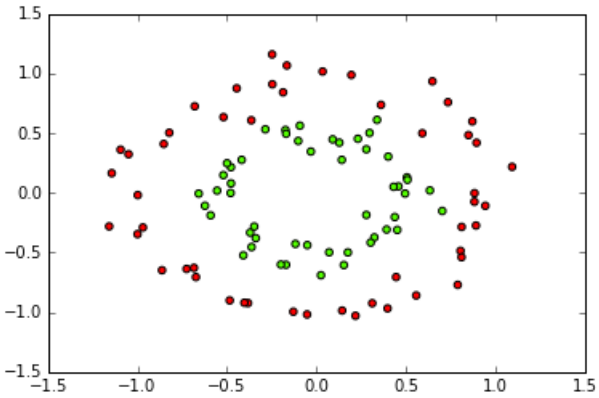
In [1]:

```
import numpy as np
from sklearn.datasets import make_circles
X, y = make_circles(noise=.1, factor=.5)
print "X.shape:", X.shape
print "unique labels: ", np.unique(y)
```

```
X.shape: (100, 2)
unique labels:  [0 1]
```

In [2]:

```
import pylab as plt
plt.prism()  # this sets a nice color map
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.show()
```



In [3]:

```
X_train = X[:50]
y_train = y[:50]
X_test = X[50:]
y_test = y[50:]
```

In [4]:

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

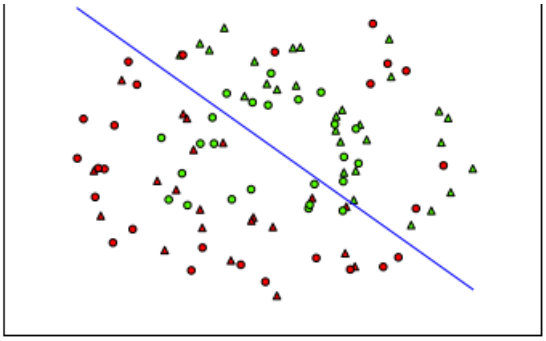
Out[4]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0)
```

In [7]:

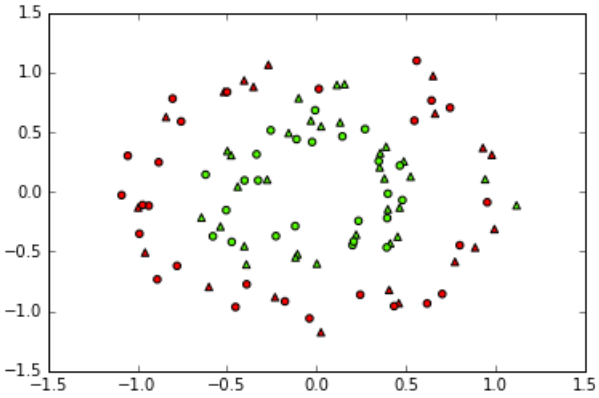
```
def plot_decision_boundary(clf, X):
    w = clf.coef_.ravel()
    a = -w[0] / w[1]
    xx = np.linspace(np.min(X[:, 0]), np.max(X[:, 0]))
    yy = a * xx - clf.intercept_ / w[1]
    plt.plot(xx, yy)
    plt.xticks(())
    plt.yticks(())
plt.prism()
y_pred_test = logreg.predict(X_test)
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred_test, marker='^')
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
plot_decision_boundary(logreg, X)
plt.xlim(-1.5, 1.5)
plt.ylim(-1.5, 1.5)
print "Accuracy of logistic regression on test set:", logreg.score(X_test, y_test)
```

```
Accuracy of logistic regression on test set: 0.42
```



In [9]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_test = knn.predict(X_test)
plt.prism() # gives us a nice color map
plt.xlim(-1.5, 1.5)
plt.ylim(-1.5, 1.5)
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred_test, marker='^')
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
plt.show()
print "Accuracy of KNN test set:", knn.score(X_test, y_test)
print "Accuracy of KNN train set:", knn.score(X_train, y_train)
```




```
Accuracy of KNN test set: 0.9
Accuracy of KNN train set: 0.96
```

In [12]:

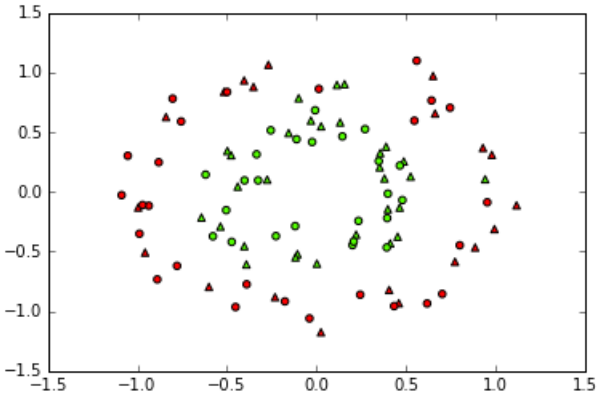
```
#use algorithm for finding best value for n_neighbors
k_range = range(1, 15)

# We can create Python dictionary using [] or dict()
scores = []

# We use a loop through the range 1 to 15
# We append the scores in the dictionary
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    scores.append(knn.score(X_train, y_train)*100)

print(scores)
#I prefer to choose n_neighbors=3
```

```
[100.0, 98.0, 98.0, 98.0, 96.0, 96.0, 94.0, 94.0, 90.0, 90.0, 76.0, 80.0, 82.0, 82.0]
```

In [15]:

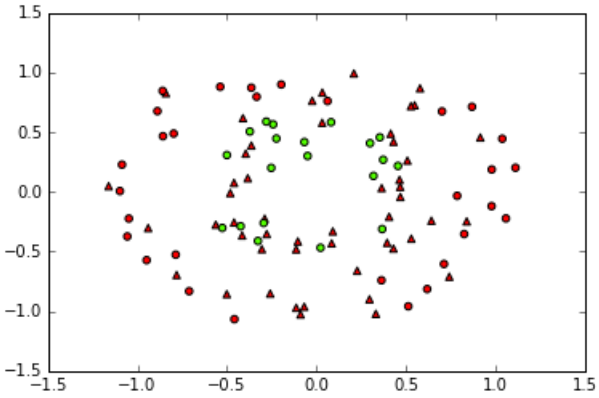
```
#LEt's test accuracy of our model
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred_test = knn.predict(X_test)
plt.prism() # gives us a nice color map
plt.xlim(-1.5, 1.5)
```

```
plt.ylim(-1.5, 1.5)
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred_test, marker='^')
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
plt.show()
print "Accuracy of KNN test set:", knn.score(X_test, y_test)*100,"%"
print "The best Accuracy of KNN train set:", knn.score(X_train, y_train)*100,"%"
```



```
Accuracy of KNN test set: 92.0 %
The best Accuracy of KNN train set: 98.0 %
```

In [95]:



```
Accuracy of KNN test set:
---------------------------------------------------------------------------
NotFittedError                            Traceback (most recent call last)
<ipython-input-95-ef6bd118ea5e> in <module>()
      5 plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
      6 plt.show()
----> 7 print "Accuracy of KNN test set:", knn.score(X_test, y_test)
      8 print "Accuracy of KNN train set:", knn.score(X_train, y_train)


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)
    318         """
    319         if self._fit_method is None:
--> 320             raise NotFittedError("Must fit neighbors before querying.")
    321
    322         if n_neighbors is None:
```

```
NotFittedError: Must fit neighbors before querying.
```

In [20]:

```
from sklearn.datasets import fetch_mldata
from sklearn.utils import shuffle
mnist = fetch_mldata("MNIST original")
X_digits, y_digits = mnist.data, mnist.target
X_digits, y_digits = shuffle(X_digits, y_digits)
```

In [29]:

```
X_digits_train = X_digits[:1000]
y_digits_train = y_digits[:1000]
X_digits_valid = X_digits[1000:2000]
y_digits_valid = y_digits[1000:2000]
X_digits_test = X_digits[2000:3000]
y_digits_test = y_digits[2000:3000]
```

In [30]:

```
#knn = KNeighborsClassifier(n_neighbors=3)
#knn.fit(X_digits_valid, y_digits_valid)
k_range = range(1, 15)

# We can create Python dictionary using [] or dict()
scores = []

# We use a loop through the range 1 to 15
# We append the scores in the dictionary
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_digits_train, y_digits_train)
    scores.append(knn.score(X_digits_test, y_digits_test)*100)

print(scores)
#print "Accuracy of KNN train set:", knn.score(X_digits_train, y_digits_train)*100,"%"
#print "Accuracy of KNN validation set:", knn.score(X_digits_valid, y_digits_valid)*100,"%"
#print "Accuracy of KNN test set:", knn.score(X_digits_test, y_digits_test)*100,"%"
```

```
[89.600000000000009, 86.900000000000006, 88.900000000000006, 88.200000000000003, 88.200000000000003, 8
7.900000000000006, 87.900000000000006, 87.400000000000006, 87.0, 87.299999999999997, 86.0,
86.099999999999994, 85.700000000000003, 85.599999999999994]
```

In [31]:

```
print "Accuracy of KNN test set:",max(scores),"%"
print "Best number for k=n_neighbors is ",scores.index(max(scores))+1
```

```
Accuracy of KNN test set: 89.6 %
Best number for k=n_neighbors is  1
```

In [45]:

```
knn = KNeighborsClassifier(n_neighbors=3)
```

In [46]:

```
knn.fit(X_digits_train, y_digits_train)
```

Out[46]:

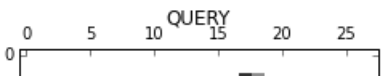
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_neighbors=3, p=2, weights='uniform')
```

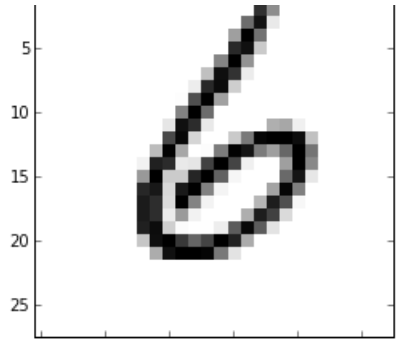
In [47]:

```
plt.rc("image", cmap="binary")
six=X_digits_train[y_digits_train==6]
plt.matshow(six[0].reshape(28, 28))
plt.title('QUERY')
```

Out[47]:

```
<matplotlib.text.Text at 0x8bee770>
```





In [48]:

```
predictions = knn.predict(X_digits_test)
prediction = knn.predict(X_digits_test)[0]
```

```
[ 0.  8.  0.  2.  9.  0.  3.  4.  2.  8.  8.  6.  3.  8.  4.  4.  6.  7.
  9.  9.  0.  9.  3.  0.  1.  4.  5.  3.  9.  7.  6.  8.  5.  2.  5.  8.
  1.  6.  3.  0.  4.  2.  1.  0.  4.  8.  7.  3.  2.  8.  5.  7.  9.  9.
  7.  3.  3.  0.  4.  5.  1.  5.  6.  7.  2.  2.  1.  8.  2.  2.  8.  4.
  0.  6.  9.  3.  8.  7.  1.  1.  5.  2.  5.  2.  0.  7.  0.  5.  4.  0.
  7.  0.  6.  3.  4.  7.  1.  0.  1.  1.  0.  8.  7.  2.  9.  3.  7.  7.
  8.  3.  3.  6.  6.  1.  6.  1.  3.  7.  2.  1.  2.  7.  9.  7.  6.  0.
  3.  0.  9.  1.  2.  1.  3.  3.  6.  6.  1.  2.  2.  2.  3.  8.  3.  6.
  6.  3.  2.  2.  7.  8.  3.  6.  2.  6.  1.  0.  9.  9.  0.  7.  3.  7.
  1.  6.  6.  5.  1.  9.  6.  5.  3.  8.  7.  7.  7.  0.  3.  9.  9.  8.
  9.  4.  7.  7.  0.  7.  0.  2.  2.  4.  4.  1.  0.  1.  7.  4.  5.  6.
  6.  3.  7.  3.  1.  6.  9.  5.  9.  1.  3.  5.  2.  7.  7.  0.  6.  2.
  3.  7.  0.  0.  3.  0.  1.  1.  3.  8.  1.  7.  9.  1.  8.  3.  8.  0.
  4.  0.  0.  6.  3.  8.  4.  4.  3.  3.  5.  7.  8.  8.  7.  2.  3.  3.
  8.  9.  2.  7.  5.  6.  0.  9.  8.  3.  3.  4.  0.  6.  2.  4.  3.  3.
  3.  3.  6.  7.  7.  4.  2.  5.  8.  2.  4.  0.  4.  7.  1.  0.  5.  9.
  0.  0.  0.  7.  2.  0.  0.  5.  4.  8.  4.  4.  3.  1.  1.  1.  5.  1.
  5.  8.  7.  5.  7.  0.  4.  7.  4.  9.  1.  0.  5.  3.  9.  5.  5.  7.
  6.  9.  4.  4.  1.  7.  2.  7.  5.  5.  3.  8.  1.  1.  9.  3.  4.  7.
  3.  1.  8.  0.  6.  1.  4.  4.  7.  4.  9.  4.  0.  6.  8.  5.  8.  6.
  3.  5.  6.  5.  3.  8.  5.  7.  9.  2.  9.  7.  1.  9.  9.  2.  2.  1.
  5.  4.  3.  2.  0.  1.  1.  9.  0.  2.  5.  0.  8.  2.  0.  3.  3.  0.
  0.  1.  5.  7.  9.  6.  8.  0.  9.  8.  1.  5.  3.  9.  8.  0.  8.  5.
  5.  3.  3.  2.  3.  7.  2.  3.  4.  0.  5.  9.  2.  9.  1.  3.  3.  0.
  6.  6.  8.  2.  1.  3.  2.  7.  8.  3.  1.  9.  7.  0.  7.  1.  1.  4.
  2.  2.  3.  0.  8.  8.  5.  3.  1.  1.  5.  0.  5.  3.  1.  5.  0.  0.
  9.  7.  2.  5.  6.  8.  7.  4.  9.  9.  1.  7.  9.  6.  1.  8.  9.  5.
  5.  2.  1.  6.  7.  3.  9.  6.  8.  7.  4.  6.  1.  4.  4.  9.  5.  6.
  3.  7.  4.  3.  0.  7.  1.  6.  9.  5.  6.  3.  2.  8.  2.  2.  0.  9.
  5.  9.  1.  0.  1.  1.  1.  9.  1.  9.  8.  6.  1.  6.  3.  0.  3.  2.
  7.  6.  8.  5.  2.  4.  1.  1.  1.  7.  7.  7.  6.  0.  9.  3.  0.  8.
  5.  5.  7.  5.  2.  9.  5.  0.  5.  2.  9.  2.  3.  6.  5.  1.  6.  1.
  2.  6.  0.  0.  2.  8.  0.  1.  3.  1.  7.  3.  1.  2.  0.  9.  9.  2.
  0.  5.  4.  0.  6.  9.  6.  0.  5.  6.  1.  9.  3.  1.  4.  1.  9.  8.
  5.  8.  7.  1.  6.  9.  4.  6.  1.  1.  0.  8.  0.  3.  5.  4.  2.  9.
  0.  6.  7.  2.  8.  0.  1.  3.  3.  1.  2.  3.  3.  5.  4.  1.  8.  5.
  1.  2.  6.  9.  7.  1.  4.  9.  5.  1.  9.  4.  1.  5.  5.  1.  8.  9.
  4.  4.  6.  9.  6.  4.  8.  0.  2.  1.  5.  1.  8.  1.  2.  6.  6.  5.
  9.  2.  2.  2.  2.  0.  6.  6.  7.  6.  1.  6.  3.  4.  3.  5.  1.  2.
  3.  2.  4.  6.  1.  1.  3.  3.  1.  7.  2.  9.  9.  0.  5.  3.  6.  6.
  0.  4.  9.  4.  0.  2.  8.  5.  6.  4.  1.  2.  1.  6.  8.  3.  1.  0.
  0.  5.  7.  1.  1.  2.  7.  8.  3.  1.  8.  1.  6.  6.  1.  1.  1.  1.
  1.  9.  1.  2.  8.  2.  5.  4.  3.  4.  3.  0.  7.  7.  0.  9.  8.  1.
  4.  2.  1.  0.  1.  4.  2.  8.  1.  9.  0.  9.  1.  1.  0.  9.  7.  0.
  6.  0.  0.  9.  7.  3.  7.  3.  7.  1.  2.  2.  7.  4.  9.  5.  9.  7.
  3.  9.  0.  3.  0.  1.  8.  0.  5.  3.  6.  5.  2.  2.  4.  3.  7.  1.
  3.  4.  9.  1.  1.  3.  1.  6.  1.  6.  2.  7.  2.  4.  3.  9.  6.  3.
  5.  8.  2.  1.  7.  6.  7.  7.  3.  9.  8.  6.  7.  8.  4.  1.  7.  4.
  8.  4.  8.  8.  7.  4.  6.  2.  0.  4.  6.  8.  1.  4.  8.  0.  0.  3.
  1.  2.  5.  3.  9.  6.  0.  1.  9.  6.  1.  2.  6.  1.  3.  1.  2.  7.
  5.  0.  5.  9.  9.  6.  3.  3.  6.  4.  1.  3.  6.  1.  3.  4.  7.  2.
  6.  6.  1.  4.  3.  6.  9.  3.  4.  9.  5.  4.  2.  0.  1.  1.  4.  2.
  1.  4.  0.  0.  7.  0.  3.  9.  0.  4.  6.  1.  1.  3.  3.  2.  2.  0.
  0.  0.  1.  9.  1.  1.  3.  6.  5.  3.  4.  0.  2.  7.  4.  9.  0.  0.
  4.  7.  2.  5.  1.  0.  0.  0.  2.  3.  2.  1.  3.  6.  2.  8.  5.  6.
  0.  5.  1.  5.  0.  3.  1.  1.  2.  2.]
```

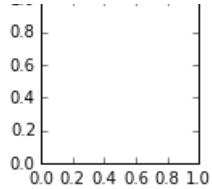
In [43]:

```
[ 0.  8.  0.  2.  9.  0.  3.  4.  2.  8.  8.  6.  3.  8.  4.  4.  6.  7.
  9.  9.  0.  9.  3.  0.  1.  4.  5.  3.  9.  7.  6.  8.  5.  2.  5.  8.
  1.  6.  3.  0.  4.  2.  1.  0.  4.  8.  7.  3.  2.  8.  5.  7.  9.  9.
  7.  3.  3.  0.  4.  5.  1.  5.  6.  7.  2.  2.  1.  8.  2.  2.  8.  4.
  0.  6.  9.  3.  8.  7.  1.  1.  5.  2.  5.  2.  0.  7.  0.  5.  4.  0.
  7.  0.  6.  3.  4.  7.  1.  0.  1.  1.  0.  8.  7.  2.  9.  3.  7.  7.
  8.  3.  3.  6.  6.  1.  6.  1.  3.  7.  2.  1.  2.  7.  9.  7.  6.  0.
  3.  0.  9.  1.  2.  1.  3.  3.  6.  6.  1.  2.  2.  2.  3.  8.  3.  6.
  6.  3.  2.  2.  7.  8.  3.  6.  2.  6.  1.  0.  9.  9.  0.  7.  3.  7.
  1.  6.  6.  5.  1.  9.  6.  5.  3.  8.  7.  7.  7.  0.  3.  9.  9.  8.
  9.  4.  7.  7.  0.  7.  0.  2.  2.  4.  4.  1.  0.  1.  7.  4.  5.  6.
  6.  3.  7.  3.  1.  6.  9.  5.  9.  1.  3.  5.  2.  7.  7.  0.  6.  2.
  3.  7.  0.  0.  3.  0.  1.  1.  3.  8.  1.  7.  9.  1.  8.  3.  8.  0.
  4.  0.  0.  6.  3.  8.  4.  4.  3.  3.  5.  7.  8.  8.  7.  2.  3.  3.
  8.  9.  2.  7.  5.  6.  0.  9.  8.  3.  3.  4.  0.  6.  2.  4.  3.  3.
  3.  3.  6.  7.  7.  4.  2.  5.  8.  2.  4.  0.  4.  7.  1.  0.  5.  9.
  0.  0.  0.  7.  2.  0.  0.  5.  4.  8.  4.  4.  3.  1.  1.  1.  5.  1.
  5.  8.  7.  5.  7.  0.  4.  7.  4.  9.  1.  0.  5.  3.  9.  5.  5.  7.
  6.  9.  4.  4.  1.  7.  2.  7.  5.  5.  3.  8.  1.  1.  9.  3.  4.  7.
  3.  1.  8.  0.  6.  1.  4.  4.  7.  4.  9.  4.  0.  6.  8.  5.  8.  6.
  3.  5.  6.  5.  3.  8.  5.  7.  9.  2.  9.  7.  1.  9.  9.  2.  2.  1.
  5.  4.  3.  2.  0.  1.  1.  9.  0.  2.  5.  0.  8.  2.  0.  3.  3.  0.
  0.  1.  5.  7.  9.  6.  8.  0.  9.  8.  1.  5.  3.  9.  8.  0.  8.  5.
  5.  3.  3.  2.  3.  7.  2.  3.  4.  0.  5.  9.  2.  9.  1.  3.  3.  0.
  6.  6.  8.  2.  1.  3.  2.  7.  8.  3.  1.  9.  7.  0.  7.  1.  1.  4.
  2.  2.  3.  0.  8.  8.  5.  3.  1.  1.  5.  0.  5.  3.  1.  5.  0.  0.
  9.  7.  2.  5.  6.  8.  7.  4.  9.  9.  1.  7.  9.  6.  1.  8.  9.  5.
  5.  2.  1.  6.  7.  3.  9.  6.  8.  7.  4.  6.  1.  4.  4.  9.  5.  6.
  3.  7.  4.  3.  0.  7.  1.  6.  9.  5.  6.  3.  2.  8.  2.  2.  0.  9.
  5.  9.  1.  0.  1.  1.  1.  9.  1.  9.  8.  6.  1.  6.  3.  0.  3.  2.
  7.  6.  8.  5.  2.  4.  1.  1.  1.  7.  7.  7.  6.  0.  9.  3.  0.  8.
  5.  5.  7.  5.  2.  9.  5.  0.  5.  2.  9.  2.  3.  6.  5.  1.  6.  1.
  2.  6.  0.  0.  2.  8.  0.  1.  3.  1.  7.  3.  1.  2.  0.  9.  9.  2.
  0.  5.  4.  0.  6.  9.  6.  0.  5.  6.  1.  9.  3.  1.  4.  1.  9.  8.
  5.  8.  7.  1.  6.  9.  4.  6.  1.  1.  0.  8.  0.  3.  5.  4.  2.  9.
  0.  6.  7.  2.  8.  0.  1.  3.  3.  1.  2.  3.  3.  5.  4.  1.  8.  5.
  1.  2.  6.  9.  7.  1.  4.  9.  5.  1.  9.  4.  1.  5.  5.  1.  8.  9.
  4.  4.  6.  9.  6.  4.  8.  0.  2.  1.  5.  1.  8.  1.  2.  6.  6.  5.
  9.  2.  2.  2.  2.  0.  6.  6.  7.  6.  1.  6.  3.  4.  3.  5.  1.  2.
  3.  2.  4.  6.  1.  1.  3.  3.  1.  7.  2.  9.  9.  0.  5.  3.  6.  6.
  0.  4.  9.  4.  0.  2.  8.  5.  6.  4.  1.  2.  1.  6.  8.  3.  1.  0.
  0.  5.  7.  1.  1.  2.  7.  8.  3.  1.  8.  1.  6.  6.  1.  1.  1.  1.
  1.  9.  1.  2.  8.  2.  5.  4.  3.  4.  3.  0.  7.  7.  0.  9.  8.  1.
  4.  2.  1.  0.  1.  4.  2.  8.  1.  9.  0.  9.  1.  1.  0.  9.  7.  0.
  6.  0.  0.  9.  7.  3.  7.  3.  7.  1.  2.  2.  7.  4.  9.  5.  9.  7.
  3.  9.  0.  3.  0.  1.  8.  0.  5.  3.  6.  5.  2.  2.  4.  3.  7.  1.
  3.  4.  9.  1.  1.  3.  1.  6.  1.  6.  2.  7.  2.  4.  3.  9.  6.  3.
  5.  8.  2.  1.  7.  6.  7.  7.  3.  9.  8.  6.  7.  8.  4.  1.  7.  4.
  8.  4.  8.  8.  7.  4.  6.  2.  0.  4.  6.  8.  1.  4.  8.  0.  0.  3.
  1.  2.  5.  3.  9.  6.  0.  1.  9.  6.  1.  2.  6.  1.  3.  1.  2.  7.
  5.  0.  5.  9.  9.  6.  3.  3.  6.  4.  1.  3.  6.  1.  3.  4.  7.  2.
  6.  6.  1.  4.  3.  6.  9.  3.  4.  9.  5.  4.  2.  0.  1.  1.  4.  2.
  1.  4.  0.  0.  7.  0.  3.  9.  0.  4.  6.  1.  1.  3.  3.  2.  2.  0.
  0.  0.  1.  9.  1.  1.  3.  6.  5.  3.  4.  0.  2.  7.  4.  9.  0.  0.
  4.  7.  2.  5.  1.  0.  0.  0.  2.  3.  2.  1.  3.  6.  2.  8.  5.  6.
  0.  5.  1.  5.  0.  3.  1.  1.  2.  2.]
```

In [7]:

```
plt.rc("image", cmap="binary") # use black/white palette for plotting
for i in xrange(3):
    plt.subplot(2,3,i+1)
    plt.imshow(neighbor[i].reshape(28,28))
plt.tight_layout()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-f07b17deef68> in <module>()
      3 for i in xrange(3):
      4     plt.subplot(2,3,i+1)
----> 5     plt.imshow(neighbor[i].reshape(28,28))
      6 plt.tight_layout()

NameError: name 'neighbor' is not defined
```



In [4]:

```
```

In [ ]:

```
```