Classification assignment with Python

Introduction

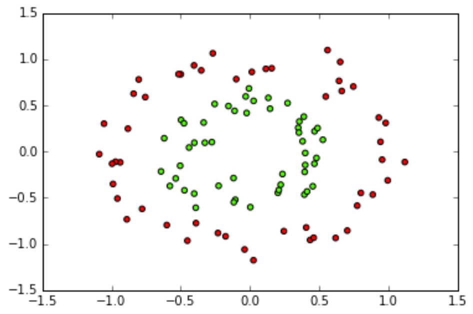
In [2]:

```
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 [3]:

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



In [4]:

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

In [6]:

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

Out[6]:

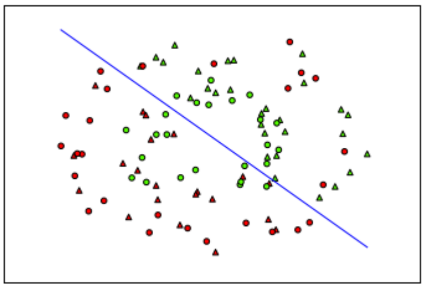
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0)
```

Comparison of logreg.score and knn.score of the same dataset

In [7]:

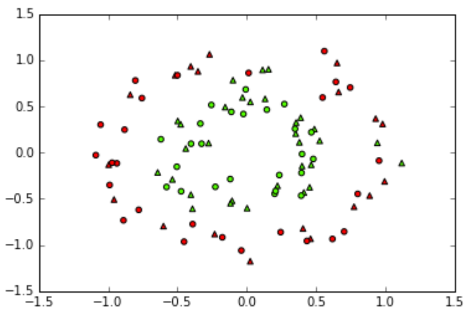
```
def plot_decision_boundary(clf, X):
    w = clf.coef_.ravel()
    a = -w[0] / w[1]
    xx = np.linspace(np.min(X[:, 0]), np.max(X[:, 0]))
    yy = a * xx - clf.intercept_ / w[1]
    plt.plot(xx, yy)
    plt.xticks(())
    plt.yticks(())
plt.prism()
y_pred_test = logreg.predict(X_test)
plt.scatter(X_test[:, 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
```

Change the n_neighbors parameter

In [12]:

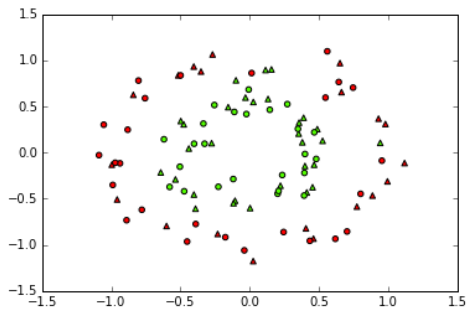
```
#use algorithm for finding best value for n_neighbors
k_range = range(1, 15)

# 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 [1]:

```
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 [2]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [3]:

```
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]
```

MNIST-Fit the model using a KNeighborsClassifier

In [9]:

```
#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,"%"
#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()
scores1 = []

# 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_valid, y_digits_valid)
    scores1.append(knn.score(X_digits_valid, y_digits_valid)*100)

print(scores1)
```

```
[87.0, 83.0, 86.299999999999997, 86.900000000000006, 86.5, 85.700000000000003, 86.5, 86.400000000000006, 86.0, 85.700000000000003,
85.0, 85.599999999999994, 85.799999999999997, 85.200000000000003]
[100.0, 95.799999999999997, 95.700000000000003, 94.299999999999997, 93.600000000000009, 92.700000000000003, 92.5,
91.700000000000003, 91.299999999999997, 91.0, 90.400000000000006, 89.5, 89.400000000000006, 88.0]
```

In [10]:

```
print "Accuracy of KNN validation set:",max(scores1),"%"
print "Best number for k=n_neighbors is ",scores1.index(max(scores1))+1
print "Accuracy of KNN test set:",max(scores),"%"
print "Best number for k=n_neighbors is ",scores.index(max(scores))+1
```

```
Accuracy of KNN validation set: 100.0 %
Best number for k=n_neighbors is  1
Accuracy of KNN test set: 87.0 %
Best number for k=n_neighbors is  1
```

NIST-Using a value of 3 for k, try to visualize the performance of the classifier on the correct predictions and in the wrong predictions

In [8]:

```
knnmodel = KNeighborsClassifier(n_neighbors=3)
knnmodel.fit(X_digits_train, y_digits_train)
predictions = knnmodel.predict(X_digits_test)
```

In [17]:

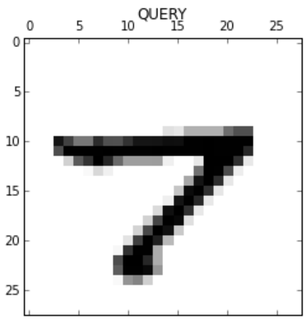
```
query=X_digits_train[y_digits_train==7]
prediction = knnmodel.predict(query)[0]
```

In [39]:

```
plt.rc("image", cmap="binary")

plt.matshow(query[0].reshape(28, 28))
plt.title('QUERY')
```

Out[39]:

```
<matplotlib.text.Text at 0x1818c030>
```

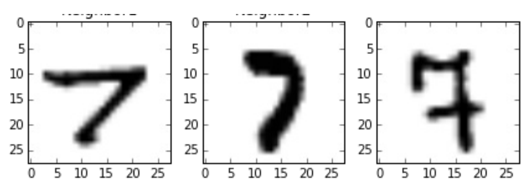


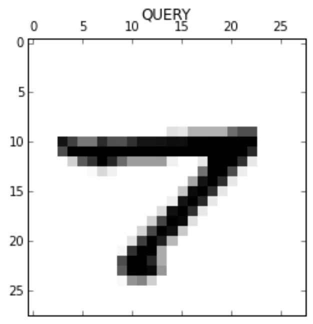
In [46]:

```
neighbor=X_digits_train[y_digits_train==prediction]
plt.rc("image", cmap="binary") # use black/white palette for plotting

for i in xrange(3):
    plt.title('Neighbor'+str(i))
    plt.subplot(2,3,i+1)
    plt.imshow(neighbor[i].reshape(28,28))

plt.tight_layout()
plt.matshow(seven[0].reshape(28, 28))
plt.title('QUERY')
```

Out[46]:

```
<matplotlib.text.Text at 0x18c0d1f0>
```



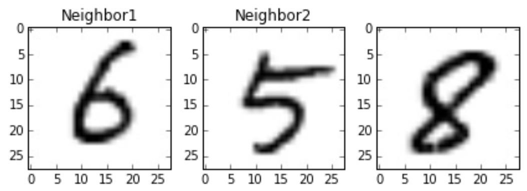


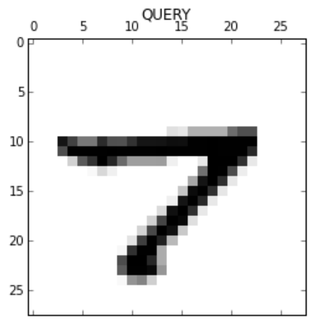
In [42]:

```
neighbor2=X_digits_train[y_digits_train!=prediction]
plt.rc("image", cmap="binary") # use black/white palette for plotting
for i in xrange(3):
    plt.title('Neighbor'+str(i))
    plt.subplot(2,3,i+1)
    plt.imshow(neighbor2[i].reshape(28,28))
plt.tight_layout()
plt.matshow(seven[0].reshape(28, 28))
plt.title('QUERY')
```

Out[42]:

```
<matplotlib.text.Text at 0x184bb8d0>
```





In [47]:

```
```

```
nan
```

In [25]:

```
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-25-ae1a05a2d7e1> in <module>()
----> 1 index(neighbor2)

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

```
In [ ]:
```

Interpretation and Conclusion

If we need to classify our dataset we should use the K-nearest neighbors algorithm. Another purpose of this exersice is a trainig of ability to choose a suitable model for our data.

For instance we used two methods (logistic regression KNN-algorithm). After fitting our log. regression model with values of train set I found the accuracy (***logreg.score***). The result was ***42%*** (0.42). It's less than ***50%,*** it means that we can't apply our model and only ***42%*** correctly defined by our model. We can notice that on the scatter plot, where some triangles of red class are situated in green class. This feature ruins accuracy of the our model. In this case we should try to use another method - ***K-nearest neighbors algorithm (knn).*** After fitting ***knn*** model with the same train set and calculating ***knn.score***, I've got accuracy ***96%*** (0.96). We can notice this on ther scatter plot, now red triangles became green and situated in green correct class. It's perfect result, but we can make it better with finding optimal value for nearest neighbors (***n_neighbors).*** For my model optimal value for ***n_neighbors*** is 3 and result of accuracy is ***98%*** (0.98). It's brilliant, because we have a high quality model, which correctly describes ***98%*** of dataset.

The last task was more interesting, because I could apply KNN algorithm for MNIST dataset. We spliited our dataset into train,test and validation tests.

I've never used before validation test, but I think it's related with cross-validation, I used validation set for estimation model properties (best value for ***k=n_neighbors.*** In my case accuracy of validation set was ***100%*** with value ***k=1,*** the same ***k=1*** was best value for train set (accuracy of train set ***87%).*** It's very good result, but sometimes when I tried to shuffle data again, I've got different results (for instance for valid set best ***k=1***, but for train set ***k=3***). It's a question. ***Does validation set estimate best value for k corectly?***

In last optional task I tried to define query from test set (for instance my query is 7) and created prediction from query. I've got similar plots with seven and "wrong" plot, where ***prediction!=7.*** Anyway I couldn't plot exactly the same plots (like in sample), but I made a attempt.

Last assignmets revealed me that we should check many models for finding the best one, unfortunatelly I tried to find best model for first task in exam, but for K-nearest I've got error: ***if y_type in ["binary", "multiclass"]:***

ValueError: continuous is not supported. I tried to fix it, but I didn't get any success. But for future now I know models such as: simple linear and multiple linear regressions, logistic regression, K-nearest neighbors classifier and clustering.