COMPSCI 371D Homework 3

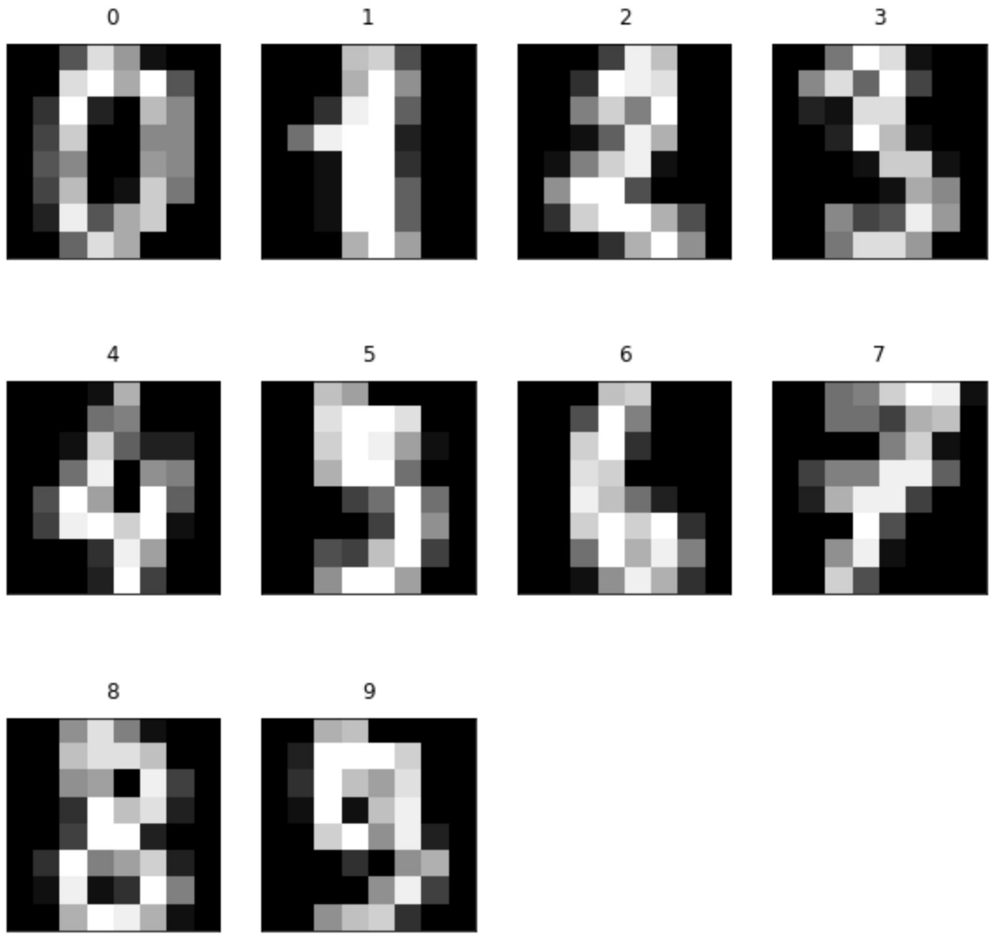
Part 1: Nearest-Neighbor Classification

In [1]:
```
from sklearn.datasets import load_digits
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

digits = load_digits()
X = digits.data
y = digits.target
image_shape = digits.images[0].shape
```

Problem 1.1

In [2]:
```
def show_sample_digits(X, y, shape):
    fig, axes = plt.subplots(3,4, figsize=(10,10))
    for r in range(len(axes)):
        for c in range(len(axes[0])):
            axs = axes[r, c]
            number = y[r*len(axes[0])+c]
            axs.matshow(X[number].reshape(8,8), cmap = plt.cm.gray)
            axs.get_xaxis().set_ticks([])
            axs.get_yaxis().set_ticks([])
            axs.set_title(number)
            if (r*len(axes[0])+c) > 9:
                axs.set_visible(False)
```

In [3]:
```
show_sample_digits(X, y, image_shape)
```



Problem 1.2

In [4]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import sklearn.metrics as met

X_train, X_test, y_train, y_test =\
    train_test_split(X, y, test_size=0.5, random_state=3)
```

In [5]:
```
predictor = KNeighborsClassifier(n_neighbors = 1)
predictor.fit(X_train, y_train)
```

Out[5]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

In [6]:
```
print('Accuracy:',round(predictor.score(X_test, y_test),4))
```

```
Accuracy: 0.9833
```

In [7]:
```
y_pred = predictor.predict(X_test)

digits = [0,1,2,3,4,5,6,7,8,9]

confusion = met.confusion_matrix(y_test, y_pred, labels = digits)
print(confusion)
```

```
[[96  0  0  0  1  0  0  0  0  0]
 [ 0 92  0  0  0  0  0  0  1  0]
 [ 0  0 99  0  0  0  0  0  0  0]
 [ 0  0  0 86  0  0  0  0  0  0]
 [ 0  1  0  0 96  0  0  0  0  0]
 [ 0  0  0  0  0 85  0  0  0  2]
 [ 0  0  0  0  0  0 77  0  0  0]
 [ 0  0  0  0  0  0  0 86  0  0]
 [ 0  2  1  3  0  0  0  0 83  3]
 [ 0  0  0  0  1  0  0  0  0 84]]
```

Problem 1.3 (Exam-Style)

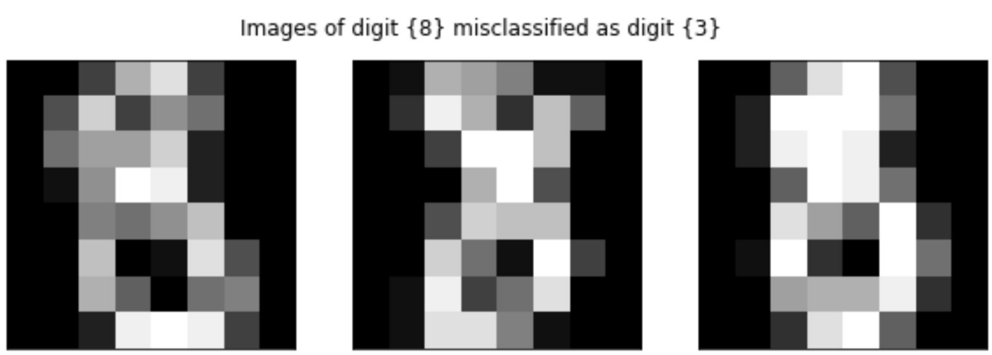
$$a = \frac{\sum_{i=1,j=1}^{K} c_{ij}}{\sum_{i=1}^{K} \sum_{j=1}^{K} c_{ij}}$$

Problem 1.4

In [8]:
```
def show_max_confusion(confusion, X, shape, y, y_hat):

    #Iterate through the confusion matrix to find the largest non-diagonal figure
    largest_error = -1
    row = -1
    col = -1
    for r in range(len(confusion)):
        for c in range(len(confusion[0])):
            if r != c:
                if largest_error < confusion[r,c]:
                    largest_error = confusion[r,c]
                    row = r
                    col = c

    #Find the Indicies of the missed predictions
    missed = []
    for i in range(len(y)):
        if (y[i] == row) and (y_hat[i] == col):
            missed.append(i)

    #Print out the Figure
    fig, axes = plt.subplots(1, len(missed), figsize=(10,10))
    fig.suptitle('Images of digit {%d} misclassified as digit {%d}' %(row, col), y
= .65)
    for n in range(len(missed)):
            axs = axes[n]
            axs.matshow(X[missed[n]].reshape(8,8), cmap = plt.cm.gray)
            axs.get_xaxis().set_ticks([])
            axs.get_yaxis().set_ticks([])

    return np.array(missed)
```

In [9]:
```
m = show_max_confusion(confusion, X_test, image_shape, y_test, y_pred)

print('The indices for the misclassified digits are')
print()
print(m)
```

```
The indices for the misclassified digits are

[106 301 641]
```



Problem 1.5

In [10]:
```
def print_nearest_distance(X, cases):
    #Finding the distance of nearest neighbors and the indices of them
    nearest_distance = []
    nearest_indices = []
    for c in cases:
        x1 = X[c]
        min_dis = 1000000
        index = -1

        for n in range(len(X)):
            if n != c:
                x2 = X[n]
                if np.linalg.norm(x1-x2) < min_dis:
                    min_dis = np.linalg.norm(x1-x2)
                    index = n

        nearest_distance.append(min_dis)
        nearest_indices.append(index)

    #Finding the next nearest neighbors
    next_dis = []
    for i in range(len(cases)):
        self = cases[i]
        nearest = nearest_indices[i]
        min_dis = 1000000
        x1 = X[self]

        for m in range(len(X)):
            if (m != self):
                if(m != nearest):
                    x2 = X[m]
                    if np.linalg.norm(x1-x2) < min_dis:
                        min_dis = np.linalg.norm(x1-x2)

        next_dis.append(min_dis)

    cell_text = []
    cell_text.append(['%.4f' % x for x in nearest_distance])
    cell_text.append(['%.4f' % x for x in next_dis])
    rows = ['Nearest Distance:', 'Next-Nearest Distance:']

    fig, ax = plt.subplots()
    ax.axis('off')
    ax.table(cellText = cell_text, rowLabels = rows, loc='center')
```

In [11]:
```
print_nearest_distance(X_test, m)
```

Nearest Distance:	23.4947	24.8193	23.6008
Next-Nearest Distance:	23.9792	25.4558	27.6225

Problem 1.6 (Exam-Style)

Since the distance can be used to decide how similar x is to one of its neighbors, so if x were to be classified, it would be classified similarly to its nearest nieghbor. If some random noise were introduced into the data set X, its distance to x could be smaller than the nearest distance, which would cause x to be misclassified, so if the nearest distance to x were decreased, the set of random noise that could be closer to x than the nearest neighbor would be reduced. This would mean that x would be more likely to be classified correctly.

Part 2: Nearest-Neighbor Regression

In [12]:
```
import pickle

with open('ames.pickle', 'rb') as f:
    data = pickle.load(f)
X, y = data['X'], data['y']
```

Problem 2.1

In [13]:
```
from sklearn.neighbors import KNeighborsRegressor

nb1 = KNeighborsRegressor(n_neighbors = 1)

nb10 = KNeighborsRegressor(n_neighbors = 10)

nb100 = KNeighborsRegressor(n_neighbors = 100)

x_space = np.array(np.linspace(0,6000,6000)).reshape(6000,1)
```

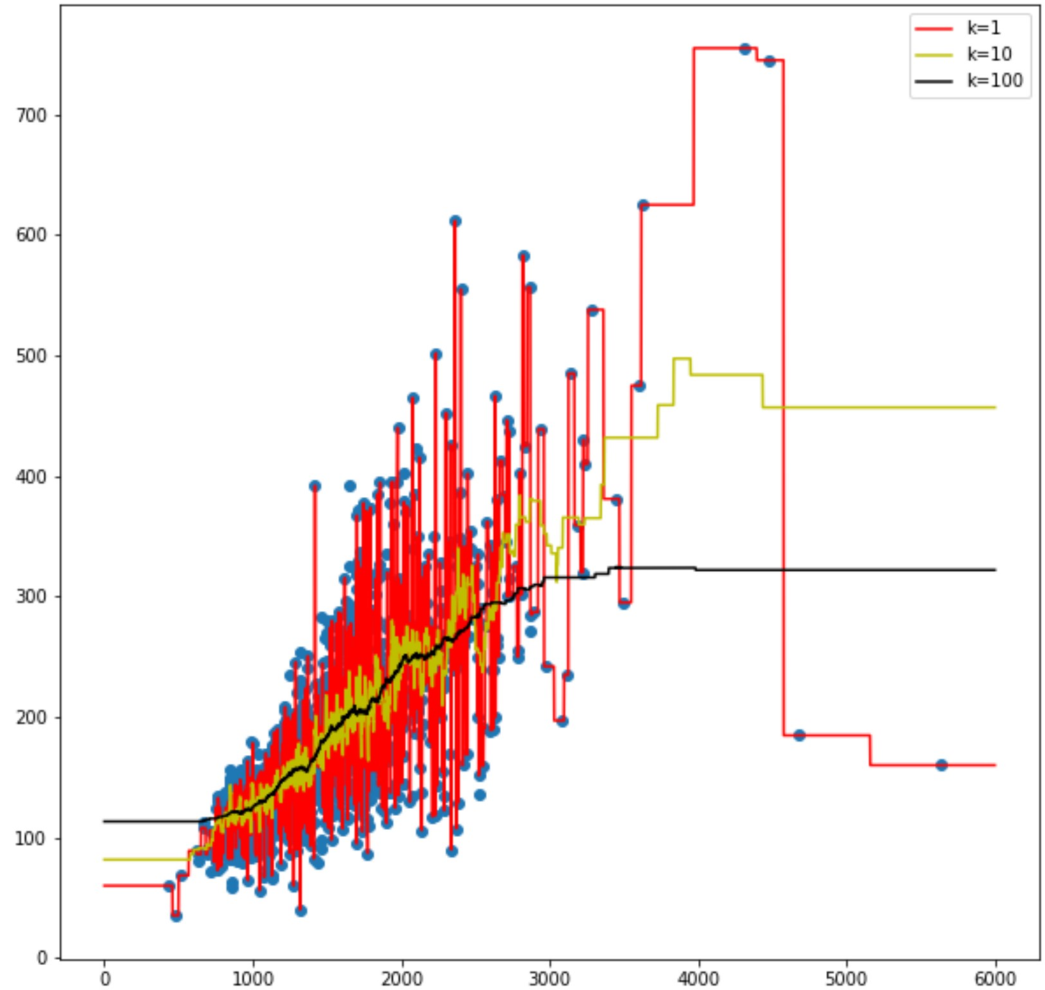
In [14]:
```
nb1.fit(X,y)

nb10.fit(X,y)

nb100.fit(X,y)
```

Out[14]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=100, p=2,
                    weights='uniform')
```

In [15]:
```
y1 = nb1.predict(x_space)

y10 = nb10.predict(x_space)

y100 = nb100.predict(x_space)
```

In [16]:
```
fig = plt.figure(figsize=(10,10))
scatter = plt.scatter(X,y)
plt.plot(x_space, y1, '-r', label = 'k=1')
plt.plot(x_space, y10, '-y', label = 'k=10')
plt.plot(x_space, y100, '-k', label = 'k=100')
plt.legend(loc='best')
```

Out[16]: <matplotlib.legend.Legend at 0x1f857aef320>



Problem 2.2 (Exam-Style)

Whenever there are outliers in a dataset, it can be risky to use the mean since the mean can be affected greatly. On the otherhand, if the dataset is evenly spaced, the median can be used since outliers won't affect the median as much.