Cognitive Algorithms - Assignment 1 (30 points)

Cognitive Algorithms
Summer semester 2018
Technische Universität Berlin
Fachgebiet Maschinelles Lernen

Due on May 9, 2018 10 am via ISIS

After completing all tasks, run the whole notebook so that the content of each cell is properly displayed. Make sure that the code was ran and the entire output (e.g. figures) is printed. Print the notebook as a PDF file and again make sure that all lines are readable - use line breaks in the Python Code '\' if necessary. Points will be deducted, if code or content is not readable!

Upload the PDF file that contains a copy of your notebook on ISIS.

Group:
Active Members:

Part 0 (0 points)

Please find the journal on ISIS https://isis.tu-berlin.de/mod/journal/view.php?id=554711 (https://isis.tu-berlin.de/mod/journal/view.php?id=554711) Each group member should fill it out individually. It asks you to give some personal information like your name, course of study and aspired degree, that we need for statistical reasons. We will not share, misuse or abuse your information. If you do not feel comfortable with sharing this information, please write an email to hannah.marienwald@campus.tu-berlin.de (mailto:hannah.marienwald@campus.tu-berlin.de). Please note, that it does not replace the registration for the exam at QISPOS or the Pruefungsamt.
Group members who did not fill out the journal or wrote an email (see above), will be assumed as inactive and deleted from the group.

Part 1: Math Recap (10 points)

The first part of this assignment is a linear algebra recap. Task 1 consists of multiple choice questions. For Task 2 you only need to write down the results and in Task 3 asks you are required to state the respective proof.

Task 1 (8 points)

Please answer questions A) to H) and check the correct answer (using an 'x'). Here is an example:
This is a question?

- [ ] wrong answer
- [ ] wrong answer
- [x] correct answer
- [ ] wrong answer

A) What is the scalar product of the following vectors $\begin{pmatrix} 1 \\ -2 \\ 0 \end{pmatrix}, \begin{pmatrix} 3 \\ 0 \\ 3 \end{pmatrix}$?

- [x] 3
- [ ] 5
- [ ] 7

B) Let $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ be two column vectors. Which of the following statements is always true?

- [x] $\mathbf{v}^T \cdot \mathbf{w} = \mathbf{w}^T \cdot \mathbf{v}$
- [ ] $\mathbf{v} \cdot \mathbf{w}^T = \mathbf{w} \cdot \mathbf{v}^T$

C) The mapping $f : \mathbb{R}^2 \ni (x, y)^\top \mapsto (x + y, y - x)^\top \in \mathbb{R}^2$ is given by the following matrix:

- [ ] $\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
- [ ] $\begin{pmatrix} 0 & 2 \\ -2 & 0 \end{pmatrix}$
- [x] $\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$

D) Which property does matrix multiplication **not** have?

- [ ] Associativity: $(AB)C = A(BC)$
- [x] Commutativity: $AB = BA$
- [ ] Distributivity: $(A + B)C = AC + BC$

E) Let $A \in \mathbb{R}^{n \times n}$ be an invertible matrix and $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ two column vectors with $A \cdot \mathbf{v} = \mathbf{w}$. Which of the following statements is always true?

- [ ] $A = \mathbf{w} \cdot \mathbf{v}^{-1}$
- [ ] $\mathbf{v} = \mathbf{w} \cdot A^{-1}$
- [x] $\mathbf{v} = A^{-1} \cdot \mathbf{w}$

(Not clear what $\mathbf{v}^{-1}$ is supposed to be)

F) The rank of the matrix $\begin{pmatrix} 4 & 4 & 4 \\ 4 & 4 & 4 \\ 4 & 4 & 4 \end{pmatrix}$ is

- [x] 1
- [ ] 3
- [ ] 4

G) For a square $n \times n$ matrix A holds

- [ ] $rank(A) = n \Rightarrow A$ is invertible, but there are invertible A with $rank(A) \neq n$
- [ ] A is invertible $\Rightarrow rank(A) = n$, but there are A with $rank(A) = n$, which are not invertible.
- [x] $rank(A) = n \iff A$ is invertible

H) Which of the following matrices is orthogonal:

- [x] $\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$
- [ ] $\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$
- [ ] $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$

Task 2 (2 points)

Please replace '?' with the correct solution.
We consider two functions f and g which transform an input vector $\mathbf{x} = (x_1, \ldots, x_d)^T \in \mathbb{R}^d$ into scalars: $f(\mathbf{x}) = \mathbf{u}^T \mathbf{x}$, $\mathbf{u} = (u_1, \ldots, u_d)^T \in \mathbb{R}^d$ and $g(\mathbf{x}) = \mathbf{x}^T \mathbf{x}$.
Compute the gradient for f and g.

- $\nabla f(\mathbf{x}) = (\frac{\partial f(\mathbf{x})}{\partial x_1}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_d})^T = u$
- $\nabla g(\mathbf{x}) = (\frac{\partial g(\mathbf{x})}{\partial x_1}, \dots, \frac{\partial g(\mathbf{x})}{\partial x_d})^T = 2x$

Part 2: Multiple Choice Questions (4 points)

In the lecture, you learned about the perceptron and the prototype classifier, which is also called the nearest centroid classifer (NCC).
Please answer questions A) to D) and check the correct answer (using an 'x').

A) The training data for a classification task is given by $(\mathbf{x_1}, y_1), \dots, (\mathbf{x_n}, y_n) \in \mathbb{R}^d \times \mathcal{C}$ where $\mathcal{C}$ is the set of classes and ...:

- [ ] ... can be of infinite size
- [ ] ... is always defined as $\mathcal{C} = \{-1, +1\}$
- [x] ... neither of the above

B) Let $\mathbf{w}$ be the weight vector. The decision boundary is ...

- [x] orthogonal to $\mathbf{w}$
- [ ] in the same direction as $\mathbf{w}$

C) Let $\mathbf{w}$ the weight vector. Which statement is true?

- [ ] $\mathbf{w}$ and $-\mathbf{w}$ yield the exact same classification
- [x] $\mathbf{w}$ and $-\mathbf{w}$ do not yield the exact same classification

D) Let $\mathbf{w} = (1, 1)^T$ and $b = 0$. Let $\mathbf{x}_1, \dots \mathbf{x_n} \in \mathbb{R}^2$ be the training data. Let $i = \{1, \dots, n\}$ and $j \in \{1, 2\}$. Which statement is true?

- [x] all data points in the first quadrant ($x_{i,j} > 0$) are classified as $+1$
- [ ] all data points in the second quadrant ($x_{i,1} < 0$ and $x_{i,2} > 0$) are classified as $+1$
- [ ] all data points in the third quadrant ($x_{i,j} < 0$) are classified as $+1$
- [ ] all data points in the fourth quadrant ($x_{i,1} > 0$ and $x_{i,2} < 0$) are classified as $+1$

Part 3: Programming (16 points)

The linear perceptron and the NCC are linear classification methods. Given training data

$$(\mathbf{x_1}, y_1), \dots, (\mathbf{x_n}, y_n) \in \mathbb{R}^d \times \{-1, 1\}$$

their goal is to learn a weight vector $\mathbf{w}$ and a bias term b, such that each new data point $\mathbf{x} \in \mathbb{R}^d$ will be assigned the correct class label via the following function:

$$\mathbf{x} \mapsto \text{sign}(\mathbf{w}^T \cdot \mathbf{x} - b)$$

The two methods use different strategies to achieve this goal. You will programm and compare the perceptron and the prototype classifier and use them to predict handwritten digits. The task is to classify one digit against all others.
If not done yet, download the data set `usps.mat` from the ISIS web site. The data set `usps.mat` contains handwritten digits from the U.S. Postal Service data set. The data set contains 2007 images and each image consits of 256 pixels.
Below you can find some useful functions for loading the data and plotting images.

In [239]:
```
import scipy as sp
import scipy.io as io
import pylab as pl
%matplotlib inline
```

In [240]:
```
''' ---- Functions for loading and plotting the images ---- '''
def load_usps_data(fname, digit=3):
    ''' Loads USPS (United State Postal Service) data from <fname>
    Definition:  X, Y = load_usps_data(fname, digit = 3)
    Input:       fname   - string
                 digit   - optional, integer between 0 and 9, default is 3
    Output:      X       -  DxN array with N images with D pixels
                 Y       -  1D array of length N of class labels
                                1 - where picture contains the <digit>
                               -1 - otherwise
    '''
    # load the data
    data = io.loadmat(fname)
    # extract images and labels
    X = data['data_patterns']
    Y = data['data_labels']
    Y = Y[digit,:]
    return X, Y

def plot_img(a):
    ''' Plots one image
    Definition: plot_img(a)
    Input:      a - 1D array that contains an image
    '''
    a2 = sp.reshape(a,(int(sp.sqrt(a.shape[0])), int(sp.sqrt(a.shape[0]))))
    pl.imshow(a2, cmap='gray')
    pl.colorbar()
    pl.setp(pl.gca(), xticks=[], yticks=[])

def plot_imgs(X, Y):
    ''' Plots 3 images from each of the two classes
    Definition:         plot_imgs(X,Y)
    Input:       X       -  DxN array of N pictures with D pixel
                 Y       -  1D array of length N of class labels {1, -1}
    '''
    pl.figure()
    for i in sp.arange(3):
        classpos = (Y == 1).nonzero()[0]
        m = classpos[sp.random.random_integers(0, classpos.shape[0]-1)]
        pl.subplot(2,3,1+i)
        plot_img(X[:, m])
    for i in sp.arange(3):
        classneg = (Y != 1).nonzero()[0]
        m = classneg[sp.random.random_integers(0, classneg.shape[0]-1)]
        pl.subplot(2,3,4+i)
        plot_img(X[:, m])
```

A) (6 points) Implement a linear perceptron by completing the function stub `train_perceptron`. We will test three different types of update rules for the learning rate (`option` $\in \{0, 1, 2\}$).

$$\text{learning rate}(t) = \begin{cases} \frac{\eta}{1+t} & \text{if option} = 0 \\ \eta & \text{if option} = 1 \\ \eta \cdot (1+t) & \text{if option} = 3 \end{cases}$$

where t is the current iteration and η the initial value of the learning rate.

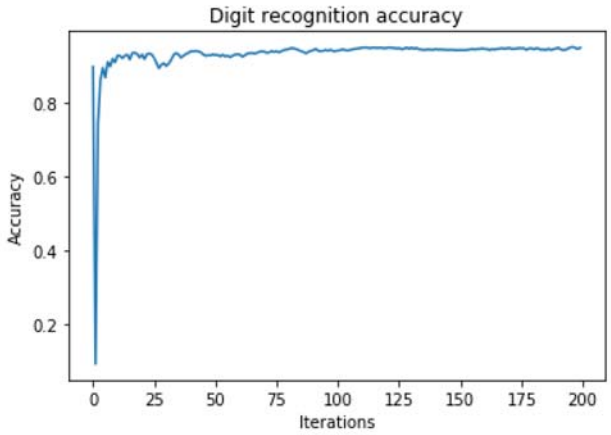
In [241]:
```
def train_perceptron(X,Y,iterations=200,eta=.1, option=0):
    ''' Trains a linear perceptron
    Definition:  w, b, acc  = train_perceptron(X,Y,iterations=200,eta=.1)
    Input:       X       -  DxN array of N data points with D features
                 Y       -  1D array of length N of class labels {-1, 1}
                 iter    -  optional, number of iterations, default 200
                 eta     -  optional, learning rate, default 0.1
                 option  -  optional, defines how eta is updated in each iteration
    Output:      w       -  1D array of length D, weight vector
                 b       -  bias term for linear classification
                 acc     -  1D array of length iter, contains classification accuracies
                            after each iteration
                            Accuracy = #correctly classified points / N
    '''
    assert option == 0 or option == 1 or option == 2
    acc = sp.zeros((iterations))
    #include the bias term by adding a row of ones to X
    X = sp.concatenate((sp.ones((1,X.shape[1])), X))
    #initialize weight vector
    weights = sp.ones((X.shape[0]))/X.shape[0]

    for it in sp.arange(iterations):

        # indices of misclassified data
        wrong = (sp.sign(weights.dot(X)) != Y).nonzero()[0]
        # compute accuracy acc[it] (1 point)
        m = wrong.shape[0]
        acc[it] = (X.shape[1] - m) / X.shape[1]
        if wrong.shape[0] > 0:

            # pick a random misclassified data point (2 points)
            mInd = wrong[sp.random.randint(0, m)]
            xm = X[:, mInd]

            #update weight vector (using different learning rates ) (each 1 point)
            if option == 0:
                L = eta / (1 + it)
                # ... your code here
            elif option == 1:
                L = eta
                # ... your code here
            elif option == 2:
                L = eta * (1 + it)
                # ... your code here

            weights = weights + L * (Y[mInd] - sp.sign(sp.dot(weights.T, xm))) * xm

    b = -weights[0]
    w = weights[1:]
    #return weight vector, bias and accuracies
    return w,b,acc

''' ------------------------------------------------------------------------------- '''
def analyse_accuracies_perceptron(digit = 3, option=0):
    ''' Loads usps.mat data and plots digit recognition accuracy in the linear perceptron
    Definition: analyse_perceptron(digit = 3)
    '''
    X,Y = load_usps_data('usps.mat',digit)
    w_per,b_per,acc = train_perceptron(X,Y, option=option)

    pl.figure()
    pl.plot(sp.arange(len(acc)),acc)
    pl.title('Digit recognition accuracy')
    pl.xlabel('Iterations')
    pl.ylabel('Accuracy')
```

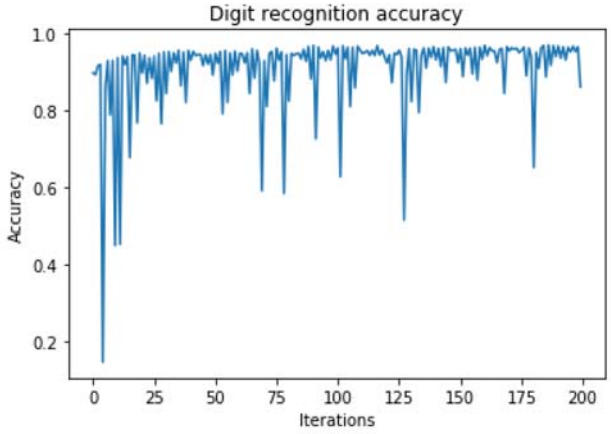
B) (3 points) Call the function `analyse_accuracies_perceptron` for a digit of your choice and all three possible `options`. It plots the classification accuracy, i.e. the percentage of correctly classified data points, as a function of iterations. Does the accuracy converge (asymptotically)? What difference do you notice for the different update rules of the learning rate? Why?

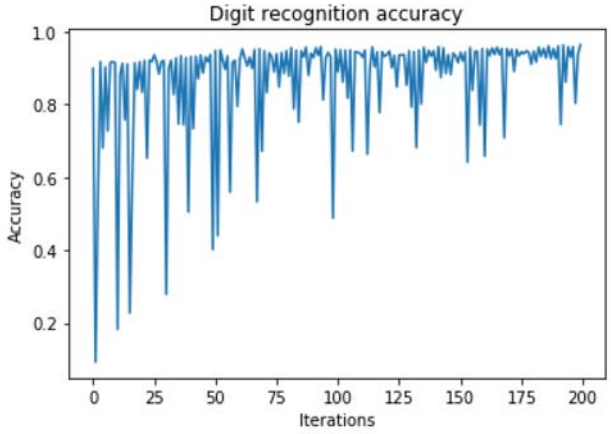
Which `option` would you prefer? Why?

Looking at the plots, it is hard to acknowledge convergence for all of them. The first update rule (which was the only one chosen sensibly) produces a well converging algorithm with good performance. The other two rules do produce good ouputs from time to time but are generally unstable (by choice obviously). Option 0 should be the preferred option here for stability and performance.

In [242]:
```
analyse_accuracies_perceptron(digit=3, option=0)
analyse_accuracies_perceptron(digit=3, option=1)
analyse_accuracies_perceptron(digit=3, option=2)
```







C) (4 points) Implement a Prototype/Nearest Centroid Classifier by completing the function stub `train_ncc` . Note that points will be deducted for the use of loops.

In [243]:
```
def train_ncc(X,Y):
    ''' Trains a prototype/nearest centroid classifier
    Definition:  w, b   = train_ncc(X,Y)
    Input:       X       -  DxN array of N data points with D features
                 Y       -  1D array of length N of class labels {-1, 1}
    Output:      w       -  1D array of length D, weight vector
                 b       -  bias term for linear classification
    '''
    # ... your code here

    # centroids
    c1 = sp.mean(X[:, Y > 0], axis = 1)
    c2 = sp.mean(X[:, Y < 0], axis = 1)

    # weight and bias
    w = c1 - c2
    b = 0.5 * sp.dot(w, w)

    return w, b
```

In [244]:
```
def plot_histogram(X, Y, w, b):
    ''' Plots a histogram of classifier outputs (w^T X) for each class with pl.hist
    The title of the histogram is the accuracy of the classification
    Accuracy = #correctly classified points / N

    Definition:     plot_histogram(X, Y, w, b)
    Input:          X       -  DxN array of N data points with D features
                    Y       -  1D array of length N of class labels
                    w       -  1D array of length D, weight vector
                    b       -  bias term for linear classification

    '''
    #Plot histogram (use pl.hist)                      +2 point (calc output, use hist)
    pl.hist((w.dot(X[:,Y<0]), w.dot(X[:,Y>0])))
    pl.xlabel("w^T X")
    pl.legend(("non-target","target"))
    #Title contains the accuracy                       +1 point (label,legend,title)
    pl.title("Acc " + str(100*sp.sum(sp.sign(w.dot(X)-b)==Y)/X.shape[1]) + "%")

''' ------------------------------------------------------------------------------- '''
def compare_classifiers(digit = 3):
    ''' Loads usps.mat data, trains the perceptron and the Nearest centroid classifiers,
    and plots their weight vector and classifier output
    Definition: compare_classifiers(digit = 3)
    '''
    X,Y = load_usps_data('usps.mat',digit)
    w_ncc,b_ncc, = train_ncc(X,Y)
    w_per,b_per,_ = train_perceptron(X,Y)

    pl.figure(figsize=(14,7))
    pl.subplot(2,2,1)
    plot_img(w_ncc)
    pl.title('NCC')
    pl.subplot(2,2,3)
    plot_histogram(X, Y, w_ncc, b_ncc)

    pl.subplot(2,2,2)
    plot_img(w_per)
    pl.title('Perceptron')
    pl.subplot(2,2,4)
    plot_histogram(X, Y, w_per, b_per)
```

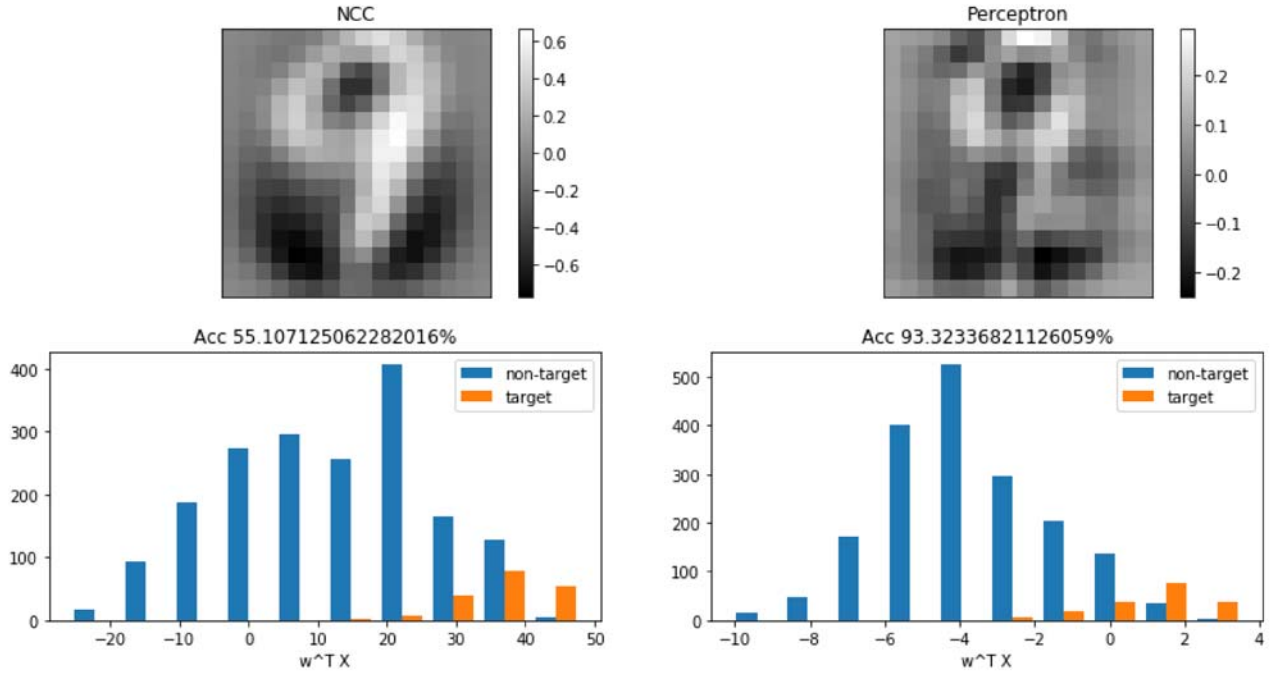
D) (3 points) Call `compare_classifiers` for a digit of your choice. It plots, for both the perceptron and the nearest centroid classifier, the histogram of classifier outputs and the weight vector.
Call the function several times for different digits. Do you notice a performance difference for the different digits? Why could this be? Show the histograms of the digits with highest difference in accuracy. Which algorithm (Nearest Centroid Classifier or Perceptron) would you prefer for this task? Why?

Hint: The function `plot_histogram` calculates the classification accuracy and plots a histogram of classifier output $\mathbf{w}^T\mathbf{x}$ for each class. To do so, X is sorted according to their labels and $w^T x$ is computed for each class. The accuracy of the algorithm is printed as the title of the plot.

The perceptron beats the NCC in every test case. This does not come as a surprise, since the perceptron is allowed to adapt to the data, whereas the NCC operates under heavy assumptiions (for example that class margins are proportional to class means margins etc.) this allows the perceptron to correctly classify more complex distributions, where the NCC yields poor results. Of course the NCC has some advantages (build performance on huge datasets should be a lot better and and the extension to multi-class classifications is more natural) but these do not come to play here. Thus, for this specific task we prefer the Perceptron for binary classification.

In [245]:
```
# ... your code for D) here

digit = 9

print ('Highest difference in accuracy with digit = 9:')
compare_classifiers(digit = digit)
```

```
Highest difference in accuracy with digit = 9:
```



We were ably to boost the performance of the perceptron slightly on average by initializing the weight vector using NCC. This also allowed for a much better peformance per Iteration, reducing the computing cost dramatically (similar performance with 1/4 of iterations). On average we observed at least a 10% drop in missclassifications. This suggests that the two techniques are not necessarily competing against one another, but should rather be used in conjunction.

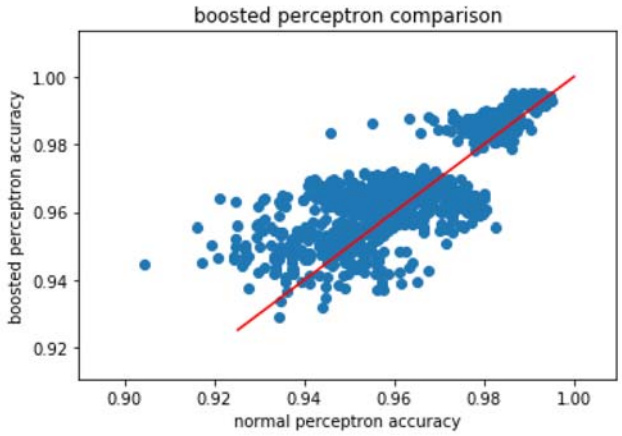
In [246]:
```
def train_perceptron_boosted(X,Y,iterations=200,eta=.1):

    c1 = sp.mean(X[:, Y > 0], axis = 1)
    c2 = sp.mean(X[:, Y < 0], axis = 1)

    w = c1 - c2
    b = 0.5 * sp.dot(w, w)

    X = sp.concatenate((sp.ones((1,X.shape[1])), X))
    weights = sp.ones((X.shape[0]))/X.shape[0]

    weights[0] = -b
    weights[1:] = w

    for it in sp.arange(iterations):

        wrong = (sp.sign(weights.dot(X)) != Y).nonzero()[0]
        miss = wrong.shape[0]

        if miss > 0:

            mInd = wrong[sp.random.randint(0, miss)]
            xm = X[:, mInd]

            weights = weights + (eta / (1 + it)) * (Y[mInd] - sp.sign(sp.dot(weights.T, xm))) * xm

    return weights[1:], -weights[0]
```

In [247]:
```
N = 100

print ('Testing perceptron vs boosted perceptron (each digit with ' + str(N) + ' iterations..')
print ('---')

acc = sp.zeros((10,N,2))

for i in range (10):

    X,Y = load_usps_data('usps.mat',i)

    for j in range(N):

        w_per,b_per,_ = train_perceptron(X,Y)
        w_per_b,b_per_b, = train_perceptron_boosted(X,Y)

        acc[i, j, 0] = sp.sum(sp.sign(w_per.dot(X)-b_per)==Y)/X.shape[1]
        acc[i, j, 1] = sp.sum(sp.sign(w_per_b.dot(X)-b_per_b)==Y)/X.shape[1]

    av_acc = sp.around(100 * sp.mean(acc, axis = 1)[i], decimals=2)
    av_acc_change = sp.around(av_acc[1] - av_acc[0], decimals = 2)
    av_less_miss = sp.around(100 * av_acc_change / (100 - av_acc[0]), decimals = 2)

    print ('Digit ' + str(i) + ': ' + str(av_acc[0]) + '% -> ' + str(av_acc[1]) + '%')
    print ("Accuracy change: " + str(av_acc_change) + '%')
    print ("Less Missclassifications: " + str(av_less_miss) + '%')
    print ('')

print ('---')
av_acc = sp.around(100 * sp.mean(sp.mean(acc, axis = 1), axis = 0), decimals=2)
av_acc_change = sp.around(av_acc[1] - av_acc[0], decimals = 2)
av_less_miss = sp.around(100 * av_acc_change / (100 - av_acc[0]), decimals = 2)
print ('Average Accuracy change: ' + str(av_acc_change)  + '%')
print ("Average Less Missclassifications: " + str(av_less_miss) + '%')

pl.scatter(acc[:,:,0].flatten(), acc[:,:,1].flatten())
pl.plot([sp.amin(measures), 1], [sp.amin(measures), 1], 'k-', color = 'r')
pl.xlabel('normal perceptron accuracy')
pl.ylabel('boosted perceptron accuracy')
pl.title('boosted perceptron comparison')
```

```
Testing perceptron vs boosted perceptron (each digit with 100 iterations..
---
Digit 0: 97.28% -> 96.24%
Accuracy change: -1.04%
Less Missclassifications: -38.24%

Digit 1: 99.09% -> 99.31%
Accuracy change: 0.22%
Less Missclassifications: 24.18%

Digit 2: 95.67% -> 96.47%
Accuracy change: 0.8%
Less Missclassifications: 18.48%

Digit 3: 96.0% -> 96.67%
Accuracy change: 0.67%
Less Missclassifications: 16.75%

Digit 4: 95.55% -> 96.32%
Accuracy change: 0.77%
Less Missclassifications: 17.3%

Digit 5: 95.46% -> 96.21%
Accuracy change: 0.75%
Less Missclassifications: 16.52%

Digit 6: 98.2% -> 98.63%
Accuracy change: 0.43%
Less Missclassifications: 23.89%
```

```
Digit 7: 98.07% -> 98.45%
Accuracy change: 0.38%
Less Missclassifications: 19.69%

Digit 8: 93.99% -> 95.02%
Accuracy change: 1.03%
Less Missclassifications: 17.14%

Digit 9: 94.67% -> 94.62%
Accuracy change: -0.05%
Less Missclassifications: -0.94%

---
Average Accuracy change: 0.39%
Average Less Missclassifications: 10.83%
```

Out[247]: `Text(0.5,1,'boosted perceptron comparison')`



In [ ]: